Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs ⋆

Nadia Busi and Roberto Gorrieri

Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy
email: `gorrieri@cs.unibo.it`

Abstract

A distributed model for the π-calculus is presented in terms of Place/Transition Petri nets with inhibitor arcs (PTI for short). Such a class of nets is equipped with a step and a causal semantics, hence allowing to study non-interleaving semantics for the π-calculus. We show the correctness of the semantics by proving that the interleaving semantics induced by the PTI semantics is fully abstract with respect to the interleaving early semantics originally defined in terms of labelled transition systems. We also argue the impossibility to define *reasonable* distributed semantics that preserve the intended non-interleaving semantics if we simply use Place/Transition nets without inhibitor arcs. Some decidability results (notably, the satisfaction of linear time μ-calculus formulae) are presented for the subclass of the π-calculus generating finite PTI nets.

1 Introduction

Non-interleaving semantics for process algebras (e.g., CCS [Mil89]) based on different classes of Petri Nets have received a lot of attention in the past (see, e.g., [DDM88,GlaVaa87,Gol90,Old91,BDK], just to mention a few). The main aim of these semantics is to offer a better understanding of the causality structure of distributed systems (useful in some cases, e.g., for error recovery) and possibly to offer some further verification techniques developed for Petri nets (e.g., net invariants).

The Place/Transition (P/T for short) net semantics proposed in the literature for CCS and related languages can be roughly classified into two main

⋆ Preliminary results reported in this paper appeared in [BusGor95] groups. The first one, which we call *location*-oriented, exploits the syntactical structure of the process terms (notably, the parallel operator) to define their associated sets of places [DDM88,Old91,DGM88]. The second one, which we call *label*-oriented, ignores the syntactical structure of the parallel operator and keeps only the information on the label of the transitions [Gol88,GorMon90]. While the former approach has the merit of being very general and has been successfully applied to several process algebras, the latter has the merit of producing small(er) net representations. Conversely, the former usually produces rather huge (1-safe P/T) nets, while the latter cannot be easily extended to cope with certain operators, e.g., restriction and alternative composition (see [GorMon95] for a discussion about this topic).

Here we show that the label-oriented approach can be extended in a natural way to powerful process algebras, by considering the prominent case of the π-calculus [MPW92], an extension of CCS for mobile processes. The basic ideas behind this extension were anticipated in [BusGor94], where we have shown how to accomodate CCS restriction and summation by exploiting P/T nets enriched with *inhibitor arcs* (PTI nets, for short). This extension is conservative in the sense that, on the sublanguage without restriction and summation, inhibitor arcs are never used. The main features of our proposal were:

(i) For each action a, there is a place named (νa) which, when containing at least one token, inhibits the execution of all the transitions with label a or $\bar{a}$.

(ii) Each agent of the form $p + q$ is first translated into the term $(\{k\}p|\{\bar{k}\}q)$, where k is called a *conflict* name and $\bar{k}$ is its contrasting conflict name. The net semantics contains a place for any conflict name k, which, when holding at least one token, inhibits all the transitions starting from places/agents decorated by $\bar{k}$ (and symmetrically for $\bar{k}$, with the assumption that $\bar{\bar{k}} = k$).

(iii) Moreover, as we do not consider the syntactical structure of parallel composition, we always need fresh names for actions to rename the restriction binders.

Hence, this net semantics implements restriction and alternative composition by checking whether the transition is inhibited by any restriction or conflict it might be subject to. In the case of the π-calculus, the main new problems one has to face are the following:

(i) the explicit presence of values for messages (which are actually names), and the fact that

(ii) restriction is a *dynamic* operator which changes the scope of application as the computation proceeds.

Both aspects are easily accommodated in our approach based on PTI nets, yielding a fully satisfactory (low-level) net-oriented semantics for a rich dialect of π-calculus. We claim that our net semantics is simple because:

(*i*) Its definition is very concise if compared with the original one, based on labelled transition systems. As a matter of fact, only five axiom schemata are needed to generate the whole set of labeled net transitions. Moreover, even if labeled, they are very much in the style of the CHAM [BerBou92], where no induction on the syntactic structure needs to be performed.

(*ii*) The intuition behind the net semantics is immediate, once one has understood the role of inhibitors. Indeed, our semantics clearly reflects the "inhibiting role" played by restriction and the nature of distributed choice implemented through inhibiting conflict names.

(*iii*) It provides concise net descriptions of both distributed choice (*linear* in the size of the components, as opposed to quadratic solutions in [DDM88,Old91]) and restriction (by not interpreting parallel composition as *disjoint* union on places as in [DDM88,Old91]).

The main results of the paper are the following:

- **Soundness.** The interleaving marking graph associated to the PTI $Net(p)$ for any π-calculus term p tightly corresponds to its *early* interleaving transition system, up to a possible change of names in extrusions. Actually, a slightly weaker result is proved: the interleaving semantics induced on nets is fully abstract with respect to the original early semantics on transition systems, i.e., $p \sim q$ iff $Net(p) \sim Net(q)$.
- **Non-interleaving Semantics.** A step semantics and a causal semantics for π-calculus are induced from the corresponding ones for PTI nets. We compare our causal semantics for π-calculus with the other approaches appeared in the literature [BorSan98,DegPri95,DegPri99,MonPis95,CatSew00]. A remarkable difference w.r.t. the (most of the) above is that our proposal is based on a model and not on syntactic conditions.
- **Impossibility result** We present some arguments in support of the impossibility to provide sensible P/T net semantics for π-calculus. In particular, we will show an example of a finite PTI net originated from a π-calculus process such that its step behaviour cannot be simulated by any finite P/T net. Intuitively, this can be explained by the observation that, during an extrusion transition, to perform a bound output of a channel y, all the sequential subprocesses inside the scope of (νy) need to be updated. This can be obtained in P/T nets only by an explicit synchronisation of all these subprocesses, thus introducing possibly wrong causes.
- **Decidability results.** Finally, we show that the PTI systems generated by π-calculus processes are of a specific form, called *primitive* in [Bus02]. For finite primitive nets some interesting results are proved in [Bus02]; in particular, the decidability of the coverability problem, (place and system) boundedness, death of a transition, the reachability problem, the deadlock problem, the liveness problem, and the satisfaction of linear time μ-calculus

[Koz83] formulae. These decidability results are useful for analysing a reasonably large subset of π-calculus that is mapped onto finite primitive nets. Such a fragment includes many infinite-state processes, as parallel composition is allowed inside recursion.

The paper is organised as follows. Section 2 contains a short overview of the π-calculus. Section 3 reports background on Petri nets, their non-interleaving semantics and the related analysis issues. Section 4 introduces the PTI based net semantics for π-calculus, together with the soundness result mentioned above. Section 5 discusses the induced step and causal semantics for π-calculus. Section 6 presents the main consequences of the net semantics: the impossibility result and the decidability results, together with two examples. Finally, in Section 7 we compare our approach with related literature, we discuss further issues and hint some future research.

2 The π-Calculus

We briefly recall the syntax and the interleaving semantics of (our variant of) the π-calculus [MPW92]. We assume the reader a bit familiar with process calculi terminology.

Let $\mathcal{N}$ be a denumerable set of names, ranged over by $a, b, \ldots, x, y, \ldots$, such that $\tau \notin \mathcal{N}$. Let $\mathcal{X}$ be a denumerable set of process variables, disjoint from $\mathcal{N} \cup \{\tau\}$, ranged over by $X, Y, Z, \ldots$. The terms are generated from names by the following grammar:

$$p ::= \mathbf{0} \mid \mu.p \mid p + p \mid p \,|\, p \mid (\nu x)p \mid X \mid recX.p$$

where $\mathbf{0}$ is the terminated process, $\mu.p$ is a *sequential* term where action μ (that can be either an input $x(y)$, an output $\bar{x}y$ or a silent move τ) is first performed and then p is ready, $p+q$ is the alternative composition of processes p and q, $p \,|\, q$ is the parallel composition of p and q, $(\nu x)p$ is process p where the name x is made private (restriction), X is a process variable and $recX.p$ is the usual recursion construct.

The set $\mathcal{P}$ of *processes* contains those terms which are, w.r.t. process variables, *closed* (all the variables occur in a *rec* binder) and *guarded* (all the variables occur in a sequential subprocess). With abuse of notation, $\mathcal{P}$ will be ranged over by $p, q, \ldots$. We called *closed systems* those processes where all the names are restricted. The sets of names, free names and bound names of a process p, called $n(p)$, $fn(p)$ and $bn(p)$ respectively, are defined as usual. The definitions of substitution and alpha conversion are standard.

(Tau) $\tau.p \xrightarrow{\tau} p$

(In) $x(z).p \xrightarrow{xy} p\{y/z\}$

(Out) $\bar{x}y.p \xrightarrow{\bar{x}y} p$

(Sum) $\dfrac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'}$

(Par) $\dfrac{p \xrightarrow{\alpha} p'}{p \,|\, q \xrightarrow{\alpha} p' \,|\, q}$ $\quad bn(\alpha) \cap fn(q) = \emptyset$

(Com) $\dfrac{p \xrightarrow{\bar{x}y} p' \qquad q \xrightarrow{xy} q'}{p \,|\, q \xrightarrow{\tau} p' \,|\, q'}$

(Close) $\dfrac{p \xrightarrow{\bar{x}(w)} p' \qquad q \xrightarrow{xw} q'}{p \,|\, q \xrightarrow{\tau} (\nu w)(p' \,|\, q')}$ $\quad w \notin fn(q)$

(Res) $\dfrac{p \xrightarrow{\alpha} p'}{(\nu y)p \xrightarrow{\alpha} (\nu y)p'}$ $\quad y \notin n(\alpha)$

(Open) $\dfrac{p \xrightarrow{\bar{x}y} p'}{(\nu y)p \xrightarrow{\bar{x}(w)} p'\{w/y\}}$ $\quad y \neq x \wedge w \notin fn((\nu y)p')$

(Rec) $\dfrac{p\{\mathrm{rec}\, X.p/X\} \xrightarrow{\alpha} p'}{\mathrm{rec}\, X.p \xrightarrow{\alpha} p'}$

Table 1
Early operational semantics (symmetric rules omitted)

The (early) operational semantics for the π-calculus is the labelled transition system $(\mathcal{P}, \mathcal{A}, \longrightarrow)$, where the states are the processes in $\mathcal{P}$, $\mathcal{A}$ is the set of labels (ranged over by α), and $\longrightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is the minimal transition relation generated by the rules listed in Table 1. The labels of the transitions are of four different kinds: the silent action τ, the input xy, the free and bound outputs $\bar{x}y$ and $\bar{x}(y)$, respectively. The free and bound names of an action α, $fn(\alpha)$ and $bn(\alpha)$, are defined as follows: $fn(\tau) = bn(\tau) = \emptyset$, $fn(\bar{x}y) = fn(xy) = \{x, y\}$, $bn(\bar{x}y) = bn(xy) = \emptyset$, $fn(\bar{x}(y)) = \{x\}$, $bn(\bar{x}(y)) = \{y\}$. The names of α are $n(\alpha) = fn(\alpha) \cup bn(\alpha)$.

Let us briefly comment the rules that are less standard. Rule (Par) has a side condition that states that p can perform α in the context of q if the bound names of α do not clash with the free names of q, in order to avoid possible capture of names in q. Rules (Open) and (Close) are better commented together: whenever a term wants to perform an output of a name y which is actually restricted, then a bound output is used to label the transition to remember that some *extrusion* (i.e., enlargement of the scope of restriction) is now possible. Note that in the target state restriction disappears and that y is replaced by a fresh name w. When, in a parallel composition, a process p performs a bound output, and the process q in parallel can perform an input on the same channel x, then communication takes place and the target process $p \,|\, q$ is restricted with the new name w, to represent that also q is now in the scope of the restriction.

Note that the semantics is called *early* because in rule (In) the name z is replaced by name y at this syntactical level, and then rule (Com) simply matches that the sent value and the chosen input value are actually the same. A different discipline, called *late* semantics, could be adopted, according to which the substitution takes place only at the level of the rule (Com). See, e.g., [SW01] for more details.

Example 1 Consider term $p = (\nu c)(recX.(\bar{a}.(\bar{c}|X))|\bar{b}.c)$, which is closed and guarded (hence p is a process). It is not difficult to see that the portion of the π-calculus transition system reachable from state p, generated by the operational rules, is infinite. As a matter of fact, an infinite sequence of $\bar{a}$ can be performed, each time reaching a state where one further sequential subprocess $\bar{c}$ is accumulated in parallel. Process p is interesting for theoretical reasons, as we will show that: (*i*) a finite primitive PTI net can be associated to it, hence inheriting all the decidability properties we will show for finite primitive PTI nets; (*ii*) no finite P/T net can be defined for p which preserves the intended step semantics, and (*iii*) no reasonable P/T net can be found for p that preserves the intended causal semantics. Process p will be discussed in Section 6.1.

Terms of the pi-calculus are equipped with an observational semantics, given in terms of a strong bisimulation-based equivalence [Mil89], where we have to be careful about the free and bound names that are involved, in order to avoid the capture of free names.

Definition 2.1 A binary relation R over the set $\mathcal{P}$ of process terms is an *early bisimulation* if $(p, q) \in R$ implies:

- if $p \xrightarrow{\alpha} p'$ and $bn(\alpha) \cap fn(p, q) = \emptyset$ there exists then q' such that $q \xrightarrow{\alpha} q'$ and $(p', q') \in R$;
- symmetrically for q derivations.

Two terms p and q are *bisimilar*, written $p \sim q$, if there exists a (early) bisimulation R such that $(p, q) \in R$. $\square$

3 P/T Petri nets and inhibitor arcs

We recall some basic notions on P/T Petri nets (i.e., without capacity constraints on places) (see, e.g., [Rei85] for an introduction). We use here a non standard notation that better suits our needs. Then, we extend P/T nets with the so-called inhibitor arcs (see, e.g., [Hack76,JK95,MonRos95,Vog97]). We adopt here the notation of [BusPin99,BusPin00].

3.1 P/T nets

Let ω be the set of natural numbers and $\omega^+ = \omega \setminus \{0\}$.

Definition 3.1 Given a set S, a *finite multiset* over S is a function $m : S \to \omega$ such that the set $dom(m) = \{s \in S \,|\, m(s) \neq 0\}$ is finite. The *multiplicity* of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathcal{M}_{fin}(S)$, is ranged over by m. A multiset m such that $dom(m) = \emptyset$ is called *empty*. The set of all finite sets over S is denoted by $\wp_{fin}(S)$. We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$. The operator $\oplus$ denotes *multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes *multiset difference*: $m \setminus m'(s) =$ if $m(s) > m'(s)$ then $m(s) - m'(s)$ else 0. The *scalar product* of a number j with a multiset m is $(j \cdot m)(s) = j \cdot (m(s))$. $\square$

Definition 3.2 A P/T net is a tuple $N = (S, T, F)$, where

- S and T are the sets of *places* and *transitions*, such that $S \cap T = \emptyset$;
- $F : (S \times T) \cup (T \times S) \to \omega$ is the *flow function*.

A P/T net is of *finite synchronization* if, for all $t \in T$, the sets $\{s \in S \mid F(s, t) > 0\}$ and $\{s \in S \mid F(t, s) > 0\}$ are finite. A P/T net is *finite* if both S and T are finite. A finite multiset over the set S of places is called a *marking*. Given a marking m and a place s, we say that the place s contains $m(s)$ *tokens*. If $F(x, y) > 0$, we say that there is an arc from x to y with weight $F(x, y)$. The *preset* of a transition t is the multiset ${}^\bullet t$ over S defined as ${}^\bullet t(s) = F(s, t)$, and represents the tokens to be "consumed"; the *postset* of t is the multiset $t^\bullet$ over S defined as $t^\bullet(s) = F(t, s)$, and represents the tokens to be "produced". A transition t is *enabled* at m, usually written as $m[t\rangle$, if ${}^\bullet t \subseteq m$. The execution of a transition t enabled at m produces the marking $m' = (m \setminus {}^\bullet t) \oplus t^\bullet$. This is usually written as $m[t\rangle m'$. A finite, non empty multiset over the set T is called a *step*. A step G is enabled at m if $m_1 \subseteq m$, where $m_1 = \bigoplus_t G(t) \cdot {}^\bullet t$. The execution of a step G enabled at m produces the marking $m' = (m \setminus m_1) \oplus m_2$, where $m_2 = \bigoplus_t G(t) \cdot t^\bullet$. This is written as $m[G\rangle m'$.

A P/T system is a tuple $N(m_0) = (S, T, F, m_0)$, where (S, T, F) is a P/T net and m_0 is a multiset over$S$, called the *initial marking*. The set of markings *reachable* from m, denoted by $[m\rangle$, is defined as the least set of markings such that

- $m \in [m\rangle$
- if $m_1 \in [m\rangle$ and, for some transition $t \in T$, $m_1[t\rangle m_2$, then $m_2 \in [m\rangle$.

We say that a marking m is reachable if m is reachable from the initial marking m_0. A P/T system is said to be *safe* if any place contains at most one token in any reachable marking, i.e. $m(s) \leq 1$ for all $s \in S$ and for all $m \in [m_0\rangle$.

A *labelled* P/T net (system) over a set *Act* of labels is a tuple (S, T, F, l) $((S, T, F, m_0, l))$, where (S, T, F) is a P/T net and $l : T \to Act$ is the labelling function. $\Box$

Remark: We will consider only nets with *finite synchronization* and with *finite initial marking*, because from a philosophical point of view, a finite marking corresponds to consider a finite amount of resources, and finite synchronization corresponds to consider events that need a finite amount of resources to happen and produce a finite amount of resources. Under these assumptions, all the (possibly infinite in number) reachable markings are finite (in size) as well. We sometimes say that nets satisfying these constraints are *physically realizable*.

Definition 3.3 The *interleaving marking graph* of N is a labelled transition system $IMG(N) = (\mathcal{M}_{fin}(S), \to, m_0)$, where m_0 is the initial state and the transition relation $\to \subseteq \mathcal{M}_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is defined by $m \xrightarrow{l(t)} m'$ iff there exists a transition $t \in T$ such that $m[t\rangle m'$. The P/T systems N_1 and N_2 are *interleaving bisimilar* ($N_1 \sim N_2$) iff there exists a strong bisimulation [Mil89] R relating (the initial states of) $IMG(N_1)$ and $IMG(N_2)$.

The *step marking graph* of N is the labelled transition system $SMG(N) = (\mathcal{M}_{fin}(S), \to, m_0)$, where $\to \subseteq \mathcal{M}_{fin}(S) \times \mathcal{M}_{fin}(Act) \times \mathcal{M}_{fin}(S)$ is defined by $m \xrightarrow{A} m'$ iff there exists a step G such that $m[G\rangle m'$ and $A = l(G)$.[1] The P/T systems N_1 and N_2 are *step bisimilar* ($N_1 \sim_{step} N_2$) iff there exists a strong bisimulation R relating (the initial states of) $SMG(N_1)$ and $SMG(N_2)$. $\Box$

[1] The labelling function l is extended to multisets of transitions in the obvious way: $l(G)(a) = k$ if $(\sum_{t_i \in dom(G).l(t_i)=a} G(t_i)) = k$.

Definition 3.4 A *P/T net with inhibitor arcs* (or PTI net, for short) is a tuple $N = (S, T, F, I)$ where

- (S, T, F) is a P/T net;
- $I \subseteq S \times T$ is the *inhibiting relation*.

The *inhibitor set* of a transition t is the set ${}^{\circ}t = \{s \in S \mid (s, t) \in I\}$, and represents the places to be "tested for absence" of tokens. This changes the definition of enabling: a transition t is enabled at m, written $m[t\rangle$, if ${}^{\bullet}t \subseteq m$ (the tokens to be consumed are available) and $dom(m) \cap {}^{\circ}t = \emptyset$ (no place in the inhibitor set of t is marked). Any transition t for which ${}^{\circ}t \cap dom({}^{\bullet}t) \neq \emptyset$ can never fire, thus it is called *blocked*. The execution of a transition t enabled at m producing the marking m', written $m[t\rangle m'$, is defined as for P/T nets.

We say that s is an *inhibiting place* if there exists a transition t such that $s \in {}^{\circ}t$. We denote with $Inhib(N)$ the set of all the inhibiting places of net N.

A PTI system is a tuple $N(m_0) = (S, T, F, I, m_0)$, where (S, T, F, I) is a PTI net and m_0 is a multiset over S, called the *initial marking*. A labelled PTI net is said to be *transition simple* if its transitions are completely determined by their label and arcs, i.e., for all $t, u \in T$, $l(t) = l(u)$, ${}^{\bullet}t = {}^{\bullet}u$, $t^{\bullet} = u^{\bullet}$ and ${}^{\circ}t = {}^{\circ}u$ implies $t = u$. If a labelled PTI net is transition simple, we can adopt the following alternative definition, which is equivalent to the standard one: we define a net as a tuple $N = (S, Act, T)$, where S is the set of places, Act is the set of transition labels and $T \subseteq \mathcal{M}_{fin}(S) \times \mathcal{P}_{fin}(S) \times Act \times \mathcal{M}_{fin}(S)$ is the set of transitions. Here each transition is represented by the tuple $({}^{\bullet}t, {}^{\circ}t, l(t), t^{\bullet})$ or, more intuitively, as $({}^{\bullet}t, {}^{\circ}t) \xrightarrow{l(t)} t^{\bullet}$. □

We adopt the usual notation to draw PTI nets: places are represented as circles, transitions as segments, flow arcs as directed segments (i.e. with an arrow at one end) and tokens as black dots inside the place. We represent an inhibitor arc as a line terminating with a small circle on the transition side. Figure 1 shows a PTI system, while Figure 8 show a P/T system.

There is no general agreement about what should be the right step semantics for PTI systems. Among the various alternative proposals ([JK95,Vog97] and [BusPin99]), we will follow the one presented in [BusPin99] (in turn inspired by [MonRos95]), because this better match our intuition about the intended step semantics of π-calculus processes when modelled with PTI systems. In particular, the mechanism we adopt for implementing distributed choice is incompatible with the other proposals of step semantics. We will come back to this point in Section 5.

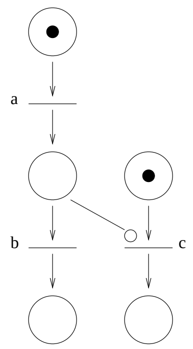

Fig. 1. A simple PTI system.

The distinguishing feature of [BusPin99] is the following property: if two transitions can happen in the same step then they can happen in either order. [2] So we have to check that (an occurrence of) a transition does not produce tokens in a place tested for absence by another one. Formally, a step G is enabled at m iff

- $m_1 \subseteq m$, where $m_1 = \bigoplus_t G(t) \cdot {}^\bullet t$
- for all $t \in dom(G)$, ${}^\circ t \cap dom(m) = \emptyset$
- for all t_1, t_2 such that $\{t_1, t_2\} \subseteq G$, we have that $dom(t_1^\bullet) \cap {}^\circ t_2 = \emptyset$

The third condition ensures that, for each pair of occurrences of transitions in the step, it never happens that one occurrence puts a token in a place inhibiting the other one. The execution of a step G enabled at m producing the marking m', written $m[G\rangle m'$, is defined as for P/T nets. In [BusPin99] it is proved that if a step G can fire, $m[G\rangle m'$, then for any linearization of its transitions $t_1, t_2, \ldots t_n$ (i.e., such that $G(t) = |\{i \mid 1 \leq i \leq n \wedge t_i = t\}|$), there exist markings m_1, m_2, m_{n-1} such that $m[t_1\rangle m_1[t_2\rangle m_2 \ldots m_{n-1}[t_n\rangle m'$.

The definition of interleaving marking graph and of step marking graph of a PTI system is the same as for P/T systems. Similarly, one can define that two PTI systems N_1 and N_2 are interleaving bisimilar when there exists a strong bisimulation relating $IMG(N_1)$ and $IMG(N_2)$, and step bisimilar if a strong bisimulation exists relating $SMG(N_1)$ and $SMG(N_2)$.

3.3 Causal semantics for PTI Systems

We hint the causal semantics for PTI systems, as proposed in [BusPin00]. The formal definitions, for the interested reader, are postponed to the Appendix.

---

[2] The reverse is not true, i.e., if two transitions happen in either order then we cannot conclude that they can occur in a step; consider, e.g., two self-loop transitions sharing one place: they can occur in either order, but they cannot occur concurrently (two tokens would be needed).

We start by summarizing briefly the leading idea behind the causal semantics on P/T systems, then we will extend it to cope with inhibitor arcs.

The basic idea is to build an unfolding of the net which keeps track of the past history. The events of a net are the occurrences of its transitions, whereas the state is represented by the current marking, i.e. the number of tokens in each place. The idea is to enrich the information about the current state by decorating each token with its history, that is the event which produced it; moreover, to distinguish between different occurrences of the same transition, we record in the state the number of occurrences of each transition that have already happened. This enriched "state" will be called configuration. The initial configuration of a net is obtained by setting at zero the occurrence numbers of each transition and decorating the tokens in the initial marking with the special symbol "*", meaning that it has not been produced by any event. When an event occurs, by looking at the histories of the tokens it consumes, we obtain information on its (immediate) causal dependencies: an event e is caused by the set of events that have produced the tokens that e consumes. Obviously, the consumption of tokens decorated with * does not add causal dependencies. To take into account the branching structure, from the set of the causal executions of a system, we construct a *causal tree* [DarDeg89] by a transitive closure of the immediate causal relation on events; then we can compare nets by means of causal bisimulation, that is bisimulation on their causal trees. In [BusPin00] it is shown that this bisimulation turns out to be as discriminating as history preserving bisimulation [RabTra88], hence no causal information is lost with respect to the one available in the classical process based semantics [Rei85].

To deal with inhibitor arcs, we extend the configurations by recording, for each place, the events that have consumed tokens from it. For an event to happen, we need to check that each inhibiting place is empty (i.e. does not contain any token). So, besides the standard causes given by the decorations of the tokens it consumes (we call *flow causes*), the event has also a set of *inhibiting causes*, given by the set of events that have removed tokens from the inhibiting places.

Causal trees [DarDeg89] are trees labelled with pairs (a, I), where a is an action and I is a set of relative pointers (implemented as natural numbers) to all the predecessors which caused the present action a. In order to record the fact that we have two different sources of causes (the flow causes originated by the transitions producing the tokens that are consumed by the transition in execution, and the inhibiting causes originated by the transitions that have removed the tokens from the places that may inhibit the transition), the causal tree we use have actually labels of the form (a, I, J). We associate a causal tree $CT(N)$ of this form to the set of the i-causal firing sequences (see the appendix) $CFS(N)$ for a net system N.

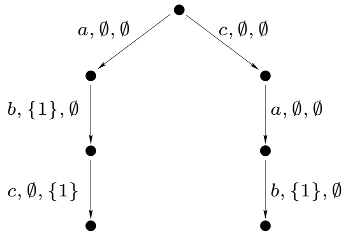

Fig. 2. The causal tree associated to the PTI system of figure 1.

Two PTI systems N_1 and N_2 are *causal bisimilar* ($N_1 \sim_c N_2$) iff there exists a bisimulation between $CT(N_1)$ and $CT(N_2)$. Actually, three different versions of causal bisimulation can be defined: (i) *mixed causal*: when in the bisimulation game we compare (a, I, J) with (a, I', J'), pretending that $I \cup J = I' \cup J'$. (ii) *distinct causal*: when in the bisimulation game we compare (a, I, J) with (a, I', J'), pretending that $I = I'$ and $J = J'$. (iii) *flow causal*: when in the bisimulation game we compare (a, I, J) with (a, I', J'), pretending that $I = I'$ and ignoring J, J'. The flow causal is actually the approach of [BorSan98], where inhibiting causes (a subset of the *object* dependencies in that paper) are ignored because interleaving equivalent processes show the same object dependencies. If not differently specified, by causal semantics we mean the mixed causal semantics.

As an example, consider the net in Figure 1. Its associated causal tree is reported in Figure 2, where it is to be observed that (i) action b is always (flow) caused by a, (ii) transition c can happen immediately before a, but not immediately after a (asymmetric conflict) and (iii) there is a computation where c is (inhibiting) caused by b (which removes the token from the inhibiting place of c).

3.4 Analysis issues in finite PTI Systems

An attractive feature of finite P/T systems is the existence of a large amount of analysis techniques, permitting to decide some properties of systems such as liveness [Hack74], deadlock [CEP95,Bus02] and reachability [Kos82,May84] of states. As the addition of inhibitor arcs makes finite P/T systems Turing equivalent [Age74], most of the techniques developed for standard finite P/T systems cannot be generalized. In [Bus02] one of the authors identified a subclass of PTI systems, called *primitive systems*, for which the systems properties above are still decidable if the net is finite.

We start by recalling the notion of primitive system, then we list the decidability results for such a class of nets. We also recall an alternative characterisation of finite primitive systems as the largest class of PTI systems whose interleaving behaviour can be simulated by a finite P/T system. Finally, we show that in general the step behaviour of finite primitive systems cannot be reproduced by finite P/T systems.

Intuitively, in a primitive system it is possible to associate a limit to each inhibiting place, in such a way that, if the number of tokens in the place exceeds the limit at some stage of the computation, then that place cannot be emptied any more and hence it cannot be tested for absence of tokens by any executable transition.

Definition 3.5 A PTI system $N = (S, T, F, I, m_0)$ is *primitive* if we can compute a function $EL : Inhib(N) \to \omega$ such that

$\forall s \in Inhib(N)\, \forall m \in [m_0\rangle$
$(m(s) > EL(s) \;\Rightarrow\; \forall m' \in [m\rangle \forall t \in T\; (m'[t\rangle \Rightarrow s \notin {}^{\circ}t)).$

Given an inhibiting place s, we call $EL(s)$ the *emptiness limit* of s.[3] □

For instance, the PTI nets in Figure 1 is primitive with emptiness limit 1 for the only inhibiting place. Similarly for the net in Figure 3. Observe that if the inhibitor arc goes instead from the preset of c to transition b, then the obtained net is not primitive.

The coverability tree construction [KarMil69] has been extended in [Bus02] to finite primitive systems, thus permitting to decide the following properties: place boundedness, existence of dead transitions, coverability.

An alternative analysis technique for primitive systems, permitting to decide reachability, liveness and deadlock properties, consists in reducing a property of a finite primitive system N to a decidable property for the corresponding finite P/T system $norm(N)$, whose construction is reported in [Bus02].

Now we introduce a notion of simulation of a primitive system by a P/T system: we define a labelling of each transition of the P/T system with a transition of the primitive system, and require the firing sequences of the P/T system to simulate the firing sequences of the primitive nets via the labelling; in other words, for any transition sequence $t_1 \ldots t_n$ of the primitive system there is a transition sequence of the P/T system that is mapped by the labelling on $t_1 \ldots t_n$; moreover, the sequence obtained applying the labelling to any transition sequence of the P/T system is a transition sequence of the

[3] Note that it is not required that $EL(s)$ is the minimal value for which the condition of primitiveness is satisfied. Therefore, in principle, for a given net N, we can find different suitable functions EL_i; hence, the emptiness limit for a place s depends on the choice of the specific EL_i, as it may be not unique.

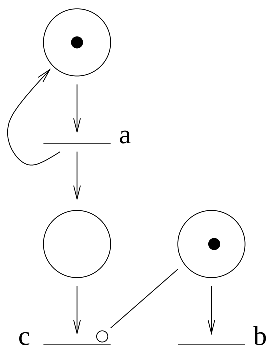

Fig. 3. A primitive system for which there exists no P/T system simulating its step transition sequences.

primitive system.

Definition 3.6 Let $N = (S, T, F, I, m_0)$ be a PTI system and $N' = (S', T', F', m'_0)$ be a P/T system. We say that N' *simulates* the transition sequences of N iff there exists a mapping $\eta : T' \to T$ such that

- if $t_1 \dots t_n$ is a transition sequence of N then there exists a transition sequence $t'_1 \dots t'_n$ of N' such that $\eta(t'_i) = t_i$ for $i = 1, \dots, n$;
- if $t'_1 \dots t'_n$ is a transition sequence of N' then $\eta(t'_1) \dots \eta(t'_n)$ is a transition sequence of N. □

Theorem 3.7 [Bus02] The class of finite primitive nets is the largest subclass of PTI systems that can be simulated by finite P/T nets. □

The above result cannot be lifted from interleaving semantics to step semantics: this is shown by providing a finite primitive system such that no finite P/T system can exhibit the same step behaviour.

Consider the system in Figure 3; it is easy to see that it is a primitive net, because the unique inhibiting place is bounded. In [Bus02] it is proved that there exists no finite P/T system with the same step transition sequences, in the following sense:

Definition 3.8 Let $N = (S, T, F, I, m_0)$ be a PTI system and $N' = (S', T', F', m'_0)$ be a P/T system. We say that N' *simulates* the step transition sequences of N iff there exists a mapping $\eta : T' \to T$ such that

- if $G_1 \dots G_n$ is a step transition sequence of N then thereexists a step transition sequence $G'_1 \dots G'_n$ of N' such that $\eta(G'_i) = G_i$ for $i = 1, \dots, n$;
- if $G'_1 \dots G'_n$ is a step transition sequence of N' then $\eta(G'_1) \dots \eta(G'_n)$ is a step transition sequence of N. □

Theorem 3.9 [Bus02] Let N be the primitive system in Figure 3. There exists no finite P/T system simulating its step transition sequences. □

The linear time μ-calculus [Koz83] is a powerful linear time logics, largely used for verification and, in [Esp97,Esp94], it was proved that the model-checking problem for finite P/T systems and closed formulas of the linear-time μ-calculus is decidable. Here we briefly recall the decidability of the linear time μ-calculus for finite primitive systems, as reported in [Bus02].

In the following we assume that *Act* is a denumerable set of symbols and that PTI systems are labeled.

Definition 3.10 Let N be a labelled PTI system. The language, ω-language and ∞-language of N are defined respectively as

- $L(N) = \{a_1 \dots a_n \mid m_0[t_1\rangle \dots [t_n\rangle m_n$ is a firing sequence of N and $l(t_i) = a_i$ for $i = 1, \dots, n\}$
- $L^\omega(N) = \{a_1 \dots a_i \dots \mid m_0[t_1\rangle \dots [t_i\rangle m_i \dots$ is an infinite firing sequence of N and $l(t_i) = a_i$ for $i \in \omega^+\}$
- $L^\infty(N) = L(N) \cup L^\omega(N)$. □

The syntax of the linear μ-calculus is the following:

$$\phi = Z \mid \neg\phi \mid \phi \wedge \phi \mid (a)\phi \mid \nu Z.\phi$$

where a ranges over a set *Act* of actions and Z over a set of propositional variables. Free and bound occurrences of variables are defined as usual. A formula is closed if no variable occurs free in it. Formulas are generated by the grammar above, and subject to the monotonicity condition that all free occurrences of a variable Z lie inside the scope of an even number of negations.

A *valuation* V of the logic maps each variable Z on a subset of Act^∞. The valuation $V[A/Z]$ is defined as $V[A/Z](Z') = \begin{cases} A & \text{if } Z' = Z \\ V(Z') & \text{otherwise} \end{cases}$

Given a word $\sigma = a_1 \dots a_i \dots$ over Act^∞, with $\sigma(1)$ we denote the first action of σ, i.e. a_1, and with σ^1 we denote the word obtained from σ by dropping the first action, i.e. $a_2 \dots a_i \dots$. The denotation of a formula consists in the set of words satisfying it. The denotation $[\![\phi]\!]_V$ of a formula ϕ according to valuation V is defined inductively as follows:

$$
\begin{aligned}
[\![Z]\!]_V &= V(Z)\\
[\![\neg\phi]\!]_V &= Act^\infty \setminus [\![\phi]\!]_V\\
[\![\phi \wedge \psi]\!]_V &= [\![\phi]\!]_V \cap [\![\psi]\!]_V\\
[\![(a)\phi]\!]_V &= \{\sigma \in Act^\infty \mid \sigma(1) = a \wedge \sigma^1 \in [\![\phi]\!]_V\}\\
[\![\nu Z.\phi]\!]_V &= \bigcup\{A \subseteq Act^\infty \mid A \subseteq [\![\phi]\!]_{V[A/Z]}\}
\end{aligned}
$$

Therefore, $[\![\nu Z.\phi]\!]_V$ is the greatest fixpoint of a function that assigns to a set A of words the set $[\![\phi]\!]_{V[A/Z]}$. The denotation of a closed formula ϕ does not depend on the valuation, hence we drop it and use the symbol $[\![\phi]\!]$. Of course, we may add derived operators, such as $\phi \vee \psi = \neg(\neg\phi \wedge \neg\phi)$. As an example of a formula in this logic, consider $\nu Z.(a)Z$, which expresses that action a is always possible. We can also express deadlock-freedom as $\nu Z. \bigvee_{a\in Act}(a)Z$.

The model checking problem for the linear time μ-calculus and finite PTI systems is defined as follows: given a finite PTI system N and a closed formula ϕ, determine if N satisfies ϕ, ie., if $L^\infty(N) \subseteq [\![\phi]\!]$. This problem is decidable for the subclass of finite primitive systems.

Theorem 3.11 [Bus02] Let N be a finite primitive system and ϕ a closed formula of the linear time μ–calculus. It is decidable if N satisfies ϕ, i.e. if $L^\infty(N) \subseteq [\![\phi]\!]$. □

The proof of this theorem is obtained by reduction to the decidability of the μ-calculus for finite P/T systems [Esp97,Esp94], as finite primitive systems can be simulated by finite P/T systems.

4 Net Semantics for the π-calculus

In this section we first describe a technique for building a PTI system for the whole π-calculus, starting from a description of its places and of its net transitions. Then we describe how to construct the subnet $dec(p)$ associated to a specific π-calculus process p, as there is no need to build the whole infinite net for the whole language if one is interested in the subnet reachable from the initial (finite) marking of $dec(p)$. Then a soundness result is provided, namely that the interleaving marking graph associated to the PTI $dec(p)$ for any π-calculus term p tightly corresponds to its *early* interleaving transition system, up to a possible change of names in extrusions. Actually, a slightly weaker result is proved: the interleaving semantics induced on nets is fully abstract with respect to the original early semantics on transition systems, i.e., $p \sim q$ iff $dec(p) \sim dec(q)$. Finally, we discuss some optimization techniques that are helpful in reducing the size of the resulting PTI net; in particular, we show that for a large fragment of the calculus (where parallel composition can occur inside recursion) the resulting net representations are finite, and we observe that under some conditions the resulting net is actually a P/T system (without inhibitor arcs).

4.1 Building the PTI system for π-calculus

We first need to define syntactic names for the places of the net we want to build. As a matter of fact, places are decorated sequential processes, where the decoration has to do with the way we handle alternative composition $+$ by means of *conflict names*; also we have a place for each conflict name and a place for each restricted name.

4.1.1 Places

Let $\mathcal{C}$ be a denumerable set of symbols disjoint from $\mathcal{N} \cup \{\tau\}$ and from $\mathcal{X}$. Let $\bar{\mathcal{C}} = \{\bar{k} \mid k \in \mathcal{C}\}$ and $Con = \mathcal{C} \cup \bar{\mathcal{C}}$. Con, called the set of conflict names, is ranged over by $k, h, \ldots$, with the assumption that $\bar{\bar{k}} = k$. A *conflict set* is a finite subset of Con, ranged over by $I, J, \ldots$; $\bar{I} = \{\bar{k} \mid k \in I\}$. When I is a singleton, we drop the set brackets for notational convenience. The infinite set S_π of places, ranged over by s (possibly indexed), is defined as follows:

$$S_\pi \;=\; \{I[\mu.p]_{\equiv_\alpha} \mid I \subseteq Con \wedge \mu.p \in \mathcal{P}\} \;\cup\; Con \;\cup\; \{(\nu x) \mid x \in \mathcal{N}\}$$

$I\mu.p$ represents a sequential process with conflict set I. This set, sometimes omitted when empty, is essential for the implementation of the distributed choice mechanism, as will be made clear in the following. Sequential processes which are alpha convertible are identified. This abstraction step is not necessary for the technical development, but it makes net description more concise. For instance, processes $x(y).(\nu z)\bar{y}z$ and $x(v).(\nu w)\bar{v}w$ are mapped to the same place $\emptyset[x(y).(\nu z)\bar{y}z]_{\equiv_\alpha}$. For notational convenience, we usually write $I\mu.p$ for $I[\mu.p]_{\equiv_\alpha}$.

Every conflict name k is a place (with the same name). k is used to prevent the execution of any transition from any sequential process $I\mu.p$ such that $\bar{k} \in I$. To better describe the implementation of the distributed choice, assume that p and q are sequential processes; $p + q$ is interpreted as the parallel composition of $\{k\}p$ and $\{\bar{k}\}q$; with the execution of an action from, say, $\{k\}p$ we produce a token in place k, hence preventing a later execution of an action from $\{\bar{k}\}q$.

Finally, observe that for any name x there is a place (νx). A token in such a place prevents the execution of input, output or bound output actions along channel x, as well as to send x as a free value or to receive x; however, it has no influence on synchronizations.

4.1.2 Markings corresponding to π-processes

In order to define the decomposition function *dec* which associates a finite multiset on S_π to each π-calculus process, we need some auxiliary definitions.

Definition 4.1 Three operators on places are defined below.

The operator $I(s)$ is defined on places as follows: $I(J\mu.p) = (I \cup J)\mu.p$, $I(k) = k$ and $I((\nu x)) = (\nu x)$.

The renaming $s\{y/x\}$ is defined on places as follows: $(I\mu.p)\{y/x\} = I(\mu.p)\{y/x\}$, $k\{y/x\} = k$, $(\nu x)\{y/x\} = (\nu y)$ and $(\nu z)\{y/x\} = (\nu z)$ if $z \neq x$.

The renaming of conflicts is defined on places as follows: $h\{k/h\} = k$, $\bar{h}\{k/h\} = \bar{k}$, $h'\{k/h\} = h'$ if $h' \notin \{h, \bar{h}\}$, $I\{k/h\} = \{h'\{k/h\} \mid h' \in I\}$, $(I\mu.p)\{k/h\} = I\{k/h\}\mu.p$ and $(\nu x)\{k/h\} = (\nu x)$. $\square$

These three operations on places can be extended to markings in the obvious way, i.e., by (multiset) elementwise application. For instance, $I(m_1 \oplus m_2) = I(m_1) \oplus I(m_2)$.

Definition 4.2 The set of conflict names, names and restricted names are defined below.

The set $c(s)$ of *conflicts of place* s is defined as: $c(I\mu.p) = \{k \in \mathcal{C} \mid k \in I \cup \bar{I}\}$, $c(k) =$ if $k \in \mathcal{C}$ then $\{k\}$ else $\{\bar{k}\}$, $c((\nu x)) = \emptyset$ and extended to markings as: $c(m) = \bigcup_{s \in dom(m)} c(s)$.

The set $n(s)$ of *names of place* s is defined as: $n(I\mu.p) = n(\mu.p)$, $n(k) = \emptyset$ and $n((\nu x)) = \{x\}$ and extended to markings as: $n(m) = \bigcup_{s \in dom(m)} n(s)$.

The set $r(m)$ of *restricted* names in marking m is defined as follows: $r(m) = \{x \mid (\nu x) \in dom(m)\}$. $\square$

For the sake of simplicity, we often write the functions above with multiple arguments, meaning the union of the applications to each argument. E.g., $r(m, m')$ means $r(m) \cup r(m')$.

We define a notion of alpha conversion on markings, which reflects the intuition that the conflicts and restricted names actually used are inessential.

Definition 4.3 The binary relation $\equiv^1_\alpha$ on markings is defined as the least

$$dec(\mathbf{0}) = \emptyset$$

$$dec(\mu.p) = \{\emptyset\mu.p\}$$

$$dec(p+q) = \{k\}(dec(p)) \oplus \{\bar{k}\}(dec(q)) \ \ k \text{ new}$$

$$dec(p \,|\, q) = dec(p) \oplus dec(q)$$

$$dec((\nu x)p) = dec(p\{z/x\}) \oplus \{(\nu z)\} \qquad z \text{ new}$$

$$dec(\mathrm{rec}\, X.p) = dec(p\{\mathrm{rec}\, X.p/X\})$$

Table 2
Decomposition function

relation generated by the two rules below:

(i) If $k, \bar{k} \notin c(m)$, then $m\{k/h\} \equiv_\alpha^1 m$.
(ii) Let $x \in r(m)$. If $w \notin n(m)$, then $m\{w/x\} \equiv_\alpha^1 m$.

Alpha conversion on markings, denoted $\equiv_\alpha$ with abuse of notation, is the reflexive and transitive closure of $\equiv_\alpha^1$. I.e., $m \equiv_\alpha m'$ if $m(\equiv_\alpha^1)^* m'$. □

As an example, consider markings $m_1 = \{(\nu y), \{k\}\bar{x}y.p, \{\bar{k}\}x(z).\mathbf{0}\}$ and $m_2 = \{(\nu w), \{h\}\bar{x}w.(p\{w/y\}), \{\bar{h}\}x(z).\mathbf{0}\}$. Clearly $m_1 \equiv_\alpha m_2$ if $w \notin n(\bar{x}y.p)$.

The function $dec : \mathcal{P} \to \mathcal{M}_{fin}(S_\pi)$, which defines the decomposition of processes into markings, is reported in Table 2. Agent $\mathbf{0}$ generates no places. The decomposition of $\mu.p$ produces one place, where $\mu.p$ has an empty set of conflicts. The case of alternative composition is interesting, as it shows that $p+q$ is turned into the multiset union of the markings for p and q, where each place of $dec(p)$ is decorated by the singleton $\{k\}$ and, symmetrically, each place in $dec(q)$ is decorated by $\{\bar{k}\}$. In order to avoid undesiderable side-effects, this conflict name k is to be chosen in such a way that it has been never used before (hence, k new). Parallel composition is interpreted as multiset union. The decomposition of a restricted process $(\nu x)p$ generates a token in a restriction place (νz) (for a new name z) and the multiset obtained from the decomposition of p where the new name z is substituted for the bound name x. Finally, a recursive process is first unwounded once and then decomposed. Guardedness on recursion variables is essential to prove the following obvious fact.

Proposition 4.4 For any π-calculus process p, $dec(p)$ is a finite multiset of places.

Note that function dec is defined up to alpha conversion of markings, because the actual choice of the *new* conflict k and of the *new* name z is inessential. Note also that a fresh conflict name k, as well as a fresh restricted name z, is to be generated for each of the *dec* applications on the right-hand-side of the transition schemata we will describe in the next section. So in a recursive term, e.g., $\mathrm{rec}\, X.x(y).\mathbf{0} + v(w).X$, there is the need for an unbounded number of fresh (conflict) names. This is a weakness of our approach that can be mitigated to some extent, as discussed in Section 4.3.

4.1.3 Net transitions

The PTI net for π-calculus, which is *transition simple* (see Def. 3.4) by construction, is the triple $N_\pi = (S_\pi, \mathcal{A}, T_\pi)$, where the infinite set T_π of net transitions is the least set generated by the axiom schemata reported in Table 3. To get a more intuitive notation, we write a transition (c, i, a, p) as $(c, i) \xrightarrow{a} p$, where c is the multiset of tokens to be consumed, i is the set of places to be tested for absence of tokens, a is the label of the transition and p is the multiset of tokens to be produced.

Axiom (tau) is trivial: if no token is present in the places of the contrasting conflict names, then the transition produces, besides the tokens of $dec(p)$, also one token in each place of the conflicts in I. In this way, any other sequential process decorated with a conflict in $\bar{I}$ is blocked forever. Axiom (in) is also very intuitive: besides the contrasting conflict names, there is to check the absence of tokens on the restriction place for the channel name x and for the received value y. Axiom (out) is very similar to the above. Axiom (open) clearly reveals the essence of bound outputs: it differs from (out) because the token in the restriction place (νy) is now to be consumed. Indeed, the effect of the extrusion of the name y is the removal of the restriction on y. Finally, axiom (sync) describes the interaction: if the two sequential processes are *compatible* (i.e., there are no contrasting conflicts, as required by the side-condition $I \cap \bar{J} = \emptyset$), then the communication can take place. Note that a *reduction semantics* (see, e.g., [SW01]) for full π-calculus can be described via axioms (tau) and (sync) only. Note also that axioms (tau) and (sync) are the only two which do not make any use of the restriction places. This means that if we are interested only to the reduction semantics, the resulting net associated to a π-calculus process is actually a P/T net (because the restriction places can be ignored – see Section 4.3 for more details). Finally, observe that conflicts are never consumed by transitions.

Example 2 Here we provide a few examples which may help in better understanding the net semantics.

- Transition $(\{\{k\}x(y).\mathbf{0}\}, \{\bar{k}, (\nu x), (\nu z)\}) \xrightarrow{xz} \{k\}$ is one of the two transitions firable from the marking $dec(x(y).\mathbf{0} + v(w).\mathbf{0})$. If it is fired, there is no way to fire the other transition, because of the generated token in place k.
- Consider process $(\nu y)(\bar{x}y.p|x(z).\mathbf{0})$, which is mapped to the marking $\{(\nu w),$

$$\text{(tau)}\quad (\{I\tau.p\},\bar{I}) \xrightarrow{\tau} dec(p) \oplus I$$

$$\text{(in)}\quad (\{Ix(z).p\},\bar{I} \cup \{(\nu x),(\nu y)\}) \xrightarrow{xy} dec(p\{y/z\}) \oplus I$$

$$\text{(out)}\quad (\{I\bar{x}y.p\},\bar{I} \cup \{(\nu x),(\nu y)\}) \xrightarrow{\bar{x}y} dec(p) \oplus I$$

$$\text{(open)}\quad (\{I\bar{x}y.p,(\nu y)\},\bar{I} \cup \{(\nu x)\}) \xrightarrow{\bar{x}(y)} dec(p) \oplus I$$

$$\text{(sync)}\quad \text{if } I \cap \bar{J} = \emptyset \text{ then}$$
$$(\{Ix(y).p, J\bar{x}z.q\},\bar{I} \cup \bar{J}) \xrightarrow{\tau} dec(p\{z/y\}) \oplus dec(q) \oplus I \oplus J$$

Table 3
Axiom schemata for (early) net transitions

$\bar{x}w.(p\{w/y\}), x(z).\mathbf{0}\}$. There are three possible enabled transitions from this marking, one for the asynchronous execution of the bound output of the left subprocesses, one for the asynchronous execution of an input from the right subprocess, and one for the synchronisation. The bound output transition is $(\{\bar{x}w.(p\{w/y\}),(\nu w)\},\{(\nu x)\}) \xrightarrow{\bar{x}(w)} dec(p\{w/y\})$. Instead the synchronisation is $(\{\bar{x}w.(p\{w/y\}), x(z).\mathbf{0}\},\emptyset) \xrightarrow{\tau} dec((p\{w/y\})\{z/w\})$. It is interesting to note that for synchronisation the restriction places play no role.

- Transition $(\{\bar{x}x.\mathbf{0},(\nu x)\},\{(\nu x)\}) \xrightarrow{\bar{x}(x)} \emptyset$ is derivable by axiom (open), so it is part of the net, but it is blocked, because it needs to consume a token from a place where the presence of a token inhibits the transition itself. In the interleaving semantics, instead, a transition from $(\nu z)\bar{z}z.\mathbf{0}$ is not derivable, because of the side condition $x \neq y$ in rule (Open). □

4.1.4 The PTI subnet associated to a π-calculus process

Given a process p, the PTI system associated to p is the subnet of N_π reachable from the initial marking $dec(p)$.

Definition 4.5 Let p be a process. The PTI system associated to p is $Net(p) = (S, \mathcal{A}, T, m_0)$, where

$$m_0 = dec(p)$$

$$S = \{s \in S_\pi \mid \exists m \in [m_0\rangle(m(s) > 0)\} \cup \{s \in Inhib(N_\pi) \mid \exists m \in [m_0\rangle\exists t(m[t\rangle \wedge s \in {}^\circ t\}$$

$$T = \{t \in T_\pi \mid \exists m \in [m_0\rangle(m[t\rangle)\}$$

It is interesting to observe that the definition above suggests a practical way of generating $Net(p)$ by means of an algorithm in least-fixpoint style. Start by $dec(p)$, which is a finite multiset, and then try to apply the five axiom schemata in order to produce the set of transitions executable from $dec(p)$ in one step. This will produce also possible new places that will be added to the current set of places. Then apply again the axiom schemata, and so on, until no new places are added and no new transitions are derivable. In order to formalize this algorithm, we need some auxiliary notations. With $m \models t$, or $m \models (c, i) \xrightarrow{a} p$, we denote the fact that transition $t = (c, i) \xrightarrow{a} p$ is derivable by means of one of the five axioms of Table 3 and that marking m enables the transition, i.e., $c \subseteq m$ and $dom(m) \cap i = \emptyset$. Formally, we can define a sequence of structures $N(q)_i = (S_i, A_i, T_i, M_i)$, for $i = 0, 1, ...$, as follows:

- $N(q)_0 = (S_0, A_0, T_0, M_0)$, where $S_0 = dom(dec(q))$, $A_0 = \emptyset$, $T_0 = \emptyset$ and $M_0 = \{dec(q)\}$.
- $N(q)_{i+1} = (S_{i+1}, A_{i+1}, T_{i+1}, M_{i+1})$, where
  · $S_{i+1} = S_i \cup \{dom(p) \cup i \mid \exists m \in M_i.m \models (c, i) \xrightarrow{a} p\}$
  · $A_{i+1} = A_i \cup \{a \mid \exists m \in M_i.m \models (c, i) \xrightarrow{a} p\}$
  · $T_{i+1} = T_i \cup \{t \mid \exists m \in M_i.m \models t\}$
  · $M_{i+1} = M_i \cup \{m' \mid \exists m \in M_i, t \in T_{i+1} \setminus T_i.m[t\rangle m'\}$.

We may define an obvious ordering $\sqsubseteq$ on such structures as follows: $N(q)_i \sqsubseteq N(q)_{i+1}$ if $S_i \subseteq S_{i+1}$ and $T_i \subseteq T_{i+1}$. This is indeed the case, hence the family $\{N(q)_i\}_i$ is a chain w.r.t. $\sqsubseteq$. Note that any component of $N(q)_i$ is finite, for any $i \in \omega$. This chain may have a finite top element if, for some i, $S_i = S_{i+1}$ and $T_i = T_{i+1}$. In such a case, we conclude that $Net(q) = (S_i, \mathcal{A}_i, T_i, m_0)$ and $Net(q)$ is a finite net.

Example 3 Consider the process $q = \bar{a}x.\mathbf{0} + \bar{b}x.\mathbf{0}$. $N(q)_0 = (S_0, A_0, T_0, M_0)$, where $S_0 = dom(dec(q)) = \{\{k\}\bar{a}x.\mathbf{0}, \{\bar{k}\}\bar{b}x.\mathbf{0}\}$, $A_0 = \emptyset$, $T_0 = \emptyset$ and $M_0 = \{dec(q)\}$. Now, transition $t_1 = (\{\{k\}\bar{a}x.\mathbf{0}\}, \{\bar{k}, (\nu a), (\nu x)\}) \xrightarrow{\bar{a}x} \{k\}$ is derivable by axiom (out) and $dec(q) \models t_1$. Similarly, transition $t_2 = (\{\{\bar{k}\}\bar{b}x.\mathbf{0}\}, \{k, (\nu b), (\nu x)\}) \xrightarrow{\bar{b}x} \{\bar{k}\}$ is derivable by axiom (out) and $dec(q) \models t_2$. Moreover, there is no other transition t_3 such that $dec(q) \models t_3$. Hence, we can conclude that $N(q)_1 = (S_1, A_1, T_1, M_1)$, where $S_1 = S_0 \cup \{k, \bar{k}, (\nu a), (\nu b), (\nu x)\}$, $A_1 = \{\bar{a}x, \bar{b}x\}$, $T_1 = \{t_1, t_2\}$ and $M_1 = \{dec(q), \{k, \{\bar{k}\}\bar{b}x.\mathbf{0}\}, \{\bar{k}, \{k\}\bar{a}x.\mathbf{0}\}\}$. Now, since no new transition is firable from any of three markings in M_1, we have done and $Net(q) = (S_1, A_1, T_1, dec(q))$. The resulting net is depicted in Figure 4.

Note also that the net semantics induces some structural axioms on π-calculus processes, i.e., if p and q are equated by the axioms, then $Net(p)$ and $Net(q)$

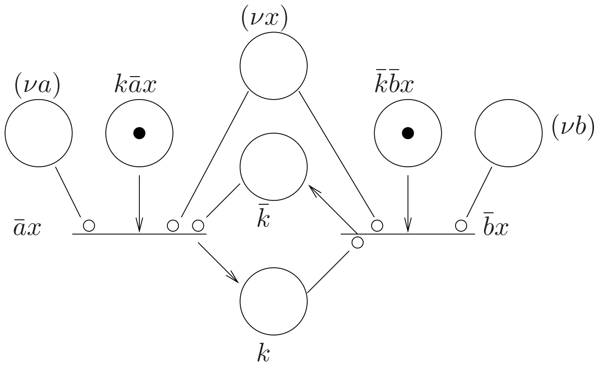

Fig. 4. The PTI system for $\bar{a}x.\mathbf{0} + \bar{b}x.\mathbf{0}$.

are isomorphic. The main axioms are for parallel composition | (it is commutative, associative, with $\mathbf{0}$ as neutral element) and for alternative composition + (commutative, but not associative). Nonetheless, it is clear that we need a more abstract semantics, possibly based on bisimulation.

We can extend to markings the definition of early bisimulation previously defined on process terms, provided that we devote some care to the treatment of bound names in our net semantics.

Definition 4.6 A binary relation R over the set of markings of N_π is an *early bisimulation* if $(m_1, m_2) \in R$ implies:

- if $m_1 \xrightarrow{\bar{x}y} m_1'$, there exists then m_2' such that $m_2 \xrightarrow{\bar{x}y} m_2'$ and $(m_1', m_2') \in R$;
- if $m_1 \xrightarrow{xy} m_1'$ and $y \notin r(m_1, m_2)$, there exists then m_2' such that $m_2 \xrightarrow{xy} m_2'$ and $(m_1', m_2') \in R$;
- if $m_1 \xrightarrow{\bar{x}(y)} m_1'$, there exist then m_2' and z such that $m_2 \xrightarrow{\bar{x}(z)} m_2'$ and – for $w \notin n(m_1, m_2)$ – $(m_1'\{w/y\}, m_2'\{w/z\}) \in R$;
- if $m_1 \xrightarrow{\tau} m_1'$, there exists then m_2' such that $m_2 \xrightarrow{\tau} m_2'$ and $(m_1', m_2') \in R$;
- symmetrically for m_2 derivations.

Two markings m_1 and m_2 are *bisimilar*, written $m_1 \sim m_2$, if there exists an early bisimulation R such that $(m_1, m_2) \in R$. □

Attention is to be paid to bound outputs. As axiom (open) uses the concrete name occuring in the output prefix (while, instead, rule (Open) for the transition system always generates a fresh, new name), we have to be careful in comparing two markings performing bound outputs: we require that $m_1'\{w/y\}$ and $m_2'\{w/z\}$ are bisimilar, where w is a new, fresh name.

The following proposition states the obvious fact that alpha-convertible markings are bisimilar.

Proposition 4.7 Let m, m' be markings. If $m \equiv_\alpha m'$ then $m \sim m'$. □

In this section, we briefly sketch the proof of retrievability of the early interleaving semantics for process p from the interleaving marking graph of the net N_π marked with the initial marking $dec(p)$.

The explicit treatment of bound names makes the correspondence a bit complex. The following proposition, rather technical but quite obvious, contains some structural properties of markings w.r.t. bisimulation equivalence, that are used in the proof of the subsequent theorems. As an example of such structural property, a marking $m \oplus j \cdot \{k\} \oplus km' \oplus \bar{k}m''$, where

- k is fresh,
- j is a positive integer,
- no name in m and m' occurs restricted in m'',
- symmetrically, no name in m'' occurs restricted in m and m',
- the intersection of the conflict names of $m \cup m'$ with m'' is empty,

is bisimilar to marking $m \oplus m'$, because $\bar{k}m''$ is blocked by the presence of j tokens in place k, no interference is possible between the names of $m \oplus m'$ and the names in m'', place k is inessential if $\bar{k}m''$ is removed, as well as the decoration k on the places in m'.

Then, two theorems compare the transition system of process p with the interleaving marking graph of $Net(p)$, showing that they are very similar. However, due to the different way rule (Open) for the transition system and axiom (open) for net deal with bound names, we cannot prove the very strong result that $p \sim dec(p)$. Nonetheless, the final corollary states that our net translation is fully abstract w.r.t. early interleaving bisimulation equivalence.

Proposition 4.8 The following hold:

(i) Let m, m' and m'' be markings. If $k, \bar{k} \notin c(m, m', m'')$, $j > 0$, $n(m, m') \cap r(m'') = \emptyset$, $r(m, m') \cap n(m'') = \emptyset$ and $c(m, m') \cap c(m'') = \emptyset$, then $m \oplus j \cdot \{k\} \oplus km' \oplus \bar{k}m'' \sim m \oplus m'$

(ii) Let m_1, m'_1, m_2 and m'_2 be markings such that $c(m_i) \cap c(m'_i) = \emptyset$, $n(m_i) \cap r(m'_i) = \emptyset$ and $r(m_i) \cap n(m'_i) = \emptyset$ for $i = 1, 2$. If $m_1 \sim m_2$ and $m'_1 \sim m'_2$, then $m_1 \oplus m'_1 \sim m_2 \oplus m'_2$.

(iii) If $x, y \notin r(dec(p))$, then $dec(p\{y/x\}) \equiv_\alpha dec(p)\{y/x\}$.

(iv) Let m and m' be markings. If $m \sim m'$, $w \notin n(m, m')$ and $x \notin r(m, m')$, then $m\{w/x\} \sim m'\{w/x\}$.

(v) Let m and m' be markings. If $m \sim m'$ and $w \notin r(m, m')$, then $m \oplus \{(\nu w)\} \sim m' \oplus \{(\nu w)\}$. $\square$

Theorem 4.9 Let p be a process.

- If $p \xrightarrow{\bar{x}y} p'$, then there exists m such that $dec(p) \xrightarrow{\bar{x}y} m$ and $m \sim dec(p')$.
- If $p \xrightarrow{xy} p'$ and $y \notin r(dec(p))$, then there exists m such that $dec(p) \xrightarrow{xy} m$ and $m \sim dec(p')$.
- If $p \xrightarrow{\bar{x}(y)} p'$, then there exist m and w such that $dec(p) \xrightarrow{\bar{x}(w)} m$ and $m \sim dec(p'\{w/y\})$.
- If $p \xrightarrow{\tau} p'$, then there exists m such that $dec(p) \xrightarrow{\tau} m$ and $m \sim dec(p')$.

Proof: By induction on the proof of transition $p \xrightarrow{\alpha} p'$. We give a sketch for the first item only (the other cases are omitted).

A transition $p \xrightarrow{\bar{x}y} p'$ can be derived by application of one of the following rules of Table 1: (Out), (Sum), (Par), (Res) or (Rec). We proceed by case analysis.

If the transition has been derived via (Out), then it is actually $\bar{x}y.p \xrightarrow{\bar{x}y} p$. Note that $dec(\bar{x}y.p)$ is the singleton $\{\bar{x}y.p\}$ and that, by application of rule (out) of Table 3, $(dec(\bar{x}y.p), \{(\nu x), (\nu y)\}) \xrightarrow{\bar{x}y} dec(p)$. Hence the thesis follows trivially.

If the transition has been derived via (Sum), then $p+q \xrightarrow{\bar{x}y} p'$ is derivable because $p \xrightarrow{\bar{x}y} p'$. By inductive hypothesis, we know that there exists m such that $dec(p) \xrightarrow{\bar{x}y} m$ and $m \sim dec(p')$. Note that $dec(p+q) = \{k\}dec(p) \oplus \{\bar{k}\}dec(q)$, with k new and no token in places k and $\bar{k}$. Of course, $\{k\}dec(p) \xrightarrow{\bar{x}y} \{k\}m \oplus \{k\}$ by application of axiom (out) of Table 3 on a suitable submarking of $\{k\}dec(p)$. Consequently, $dec(p+q) \xrightarrow{\bar{x}y} \{k\}m \oplus \{k\} \oplus \{\bar{k}\}dec(q)$. Now, property (i) of Proposition 4.8 could be applied to prove that $\{k\}m \oplus \{k\} \oplus \{\bar{k}\}dec(q)$ is bisimilar to m (and hence to $dec(p')$ as required) if the side conditions $n(m) \cap r(dec(q)) = \emptyset$, $r(m) \cap n(dec(q)) = \emptyset$ and $c(m) \cap c(dec(q)) = \emptyset$ are satisfied. Actually, we prove a slightly different result: we take a marking $m' \equiv_\alpha m$ such that the conditions above on disjointness of names w.r.t. $dec(q)$ are satisfied. Hence, $\{k\}m \oplus \{k\} \oplus \{\bar{k}\}dec(q) \sim m'$; by Proposition 4.7 $m' \sim m$, and the thesis follows by the fact that $m \sim dec(p')$.

The cases of rules (Par), (Res) and (Rec) can be proved similarly, possibly by suitably using some of the results of Proposition 4.8. $\square$

Theorem 4.10 Let p be a process.

- If $dec(p) \xrightarrow{\alpha} m$ and α is not a bound output, then there exists p' such that $p \xrightarrow{\alpha} p'$ and $m \sim dec(p')$.
- If $dec(p) \xrightarrow{\bar{x}(w)} m$, then for every $y \notin fn(p)$ there exists p' such that $p \xrightarrow{\bar{x}(y)} p'$ and $m \sim dec(p'\{w/y\})$.

Proof: By induction on (the definition of) $dec(p)$ and then by case analysis on the applicable axioms. We give a sketch of the first item only.

If $dec(p) \xrightarrow{\alpha} m$ (with α not a bound output), then the base cases are when p is one of the following: $\tau.p'$, $x(z).p'$, or $\bar{x}y.p'$ and the axiom used are (tau), (in) and (out) of Table 3, respectively. W.l.o.g., consider only $p = x(z).p'$. Then $dec(p) = \{x(z).p'\}$ and $dec(p) \xrightarrow{xy} dec(p'\{y/z\})$. By rule (In) of Table 1, $p \xrightarrow{xy} p'\{y/z\}$ and the thesis follows trivially.

Let us now consider the cases where inductive hypothesis is to be used, i.e., when $p = p_1 + p_2, p_1 \mid p_2, (\nu x)p_1$, or $\mathrm{rec}\, X.p_1$. We consider only the case $p = p_1 + p_2$. Assume $dec(p) \xrightarrow{\alpha} m$, with α not a bound output. As $dec(p) = \{k\}dec(p_1) \oplus \{\bar{k}\}dec(p_2)$ for a new conflict name k and since $dec(p_1)$ and $dec(p_2)$ are mutually exclusive via k and $\bar{k}$, either $dec(p_1) \xrightarrow{\alpha} m_1$ (with $m = \{k\}m_1 \oplus \{k\} \oplus \{\bar{k}\}dec(p_2)$), or $dec(p_2) \xrightarrow{\alpha} m_2$ (with $m = \{\bar{k}\}m_2 \oplus \{\bar{k}\} \oplus \{k\}dec(p_1)$). W.l.o.g., consider $dec(p_1) \xrightarrow{\alpha} m_1$. By inductive hypothesis, there exists p_1' such that $p_1 \xrightarrow{\alpha} p_1'$ and $m_1 \sim dec(p_1')$. Note that, by rule (Sum) of Table 1, also $p_1 + p_2 \xrightarrow{\alpha} p_1'$. So it remains to prove that $m = \{k\}m_1 \oplus \{k\} \oplus \{\bar{k}\}dec(p_2)$ is bisimilar to $dec(p_1')$, or, more simply, that $m_1 \sim m$. Now, property (i) of Proposition 4.8 could be applied to prove this, but not directly to m_1, rather to another marking $m_1' \equiv_\alpha m_1$ that satisifed all the conditions on disjointness of names listed in that proposition. Hence the thesis follows by observing that $dec(p_1') \sim m_1$ by induction, $m_1 \sim m_1'$ by Proposition 4.7, and $m_1' \sim m$ by Proposition 4.8(i).

A final remark is on the fact that the inductive argument we are using is *not* on the *structure* of the term p, rather on the defintion of function $dec(p)$. This is clear in case of $p = \mathrm{rec}\, X.p_1$, where structural induction cannot be applied. Indeed, $dec(\mathrm{rec}\, X.p_1) = dec(p\{\mathrm{rec}\, X.p/X\})$ and we can safely assume the inductive thesis holds for $dec(p\{\mathrm{rec}\, X.p/X\})$ because, by guardedness of the recursive terms, we are sure that in term $p\{\mathrm{rec}\, X.p/X\}$ the recursive binder does not occur at the top level, hence in a finite number of calls of function dec the resulting marking will be computed. $\square$

From the two theorems above, it follows immediately the following full abstraction result.

Corollary 4.11 Let p and q be processes. Then we have that $p \sim q$ if and only if $dec(p) \sim dec(q)$. $\square$

4.3 Optimization

The net semantics we have introduced is far from being optimal, meaning that there are techniques that can be implemented to get more compact net models for π-calculus processes. Here we discuss a few of these improvements and we single out a reasonably large subset of π-calculus whose processes are mapped to finite PTI systems.

The generation of an infinite system is due to the following facts:

- axiom (in) generates an infinite set of transitions exiting from the place corresponding to a sequential subprocess waiting for a message;
- the decomposition rules for choice and restriction require the generation of a fresh name; hence, if these operators lie inside a recursive definition, an infinite set of fresh names is required.

The first problem can be solved – in case the input sequential subprocess does not lie inside a recursion – by a result in [MonPis95], which says that it is sufficient to instantiate the received channel name in an input action with all the names occurring in the term plus a fresh name, to represent all the possible behaviours. Hence, this infinity problem is solved with a finite number of transitions.

The second problem cannot in general be solved: in [GolRen94] it is shown that, for a CCS process containing *unguarded* sum (i.e., the summands are not all sequential) inside recursion, no finite P/T system, respecting both the intended causal and branching behaviour, can be defined. So, we can only look for techniques that alleviate this problem in practical cases.

For instance, when the operands of a choice are all sequential processes (the so-called *guarded sum*, a limitation that is often considered sufficient for practical purposes), it is no longer necessary to introduce conflict names to model the choice: in that case, it is sufficient to glue the places corresponding to each operand in a single one. Formally, we have to do the following. We add to the syntax the guarded choice operator $\sum_{i\in I}\mu_i.p_i$, with I finite, the decomposition rule

$$dec(\sum_{i\in I}\mu_i.p_i) = \{\sum_{i\in I}\mu_i.p_i\}$$

and change the axioms in the following way:

if $\mu_i = \tau$ then

$(\{I \sum_{k\in K} \mu_k.p_k\}, \bar{I}) \xrightarrow{\tau} dec(p_i) \oplus I$

if $\mu_i = x(z)$ then

$(\{I \sum_{k\in K} \mu_k.p_k\}, \bar{I} \cup \{(\nu x), (\nu y)\}) \xrightarrow{xy} dec(p_i\{y/z\}) \oplus I$

if $\mu_i = \bar{x}y$ then

$(\{I \sum_{k\in K} \mu_k.p_k\}, \bar{I} \cup \{(\nu x), (\nu y)\}) \xrightarrow{\bar{x}y} dec(p_i) \oplus I$

if $\mu_i = \bar{x}y$ then

$(\{I \sum_{k\in K} \mu_k.p_k, (\nu y)\}, \bar{I} \cup \{(\nu x)\}) \xrightarrow{\bar{x}(y)} dec(p_i) \oplus I$

if $\mu_i = x(y), \mu'_j = \bar{x}z \wedge I \cap \bar{J} = \emptyset$ then

$(\{I \sum_{k\in K} \mu_k.p_k, J \sum_{h\in H} \mu'_h.p_h\}, \bar{I} \cup \bar{J}) \xrightarrow{\tau} dec(p_i\{z/y\}) \oplus dec(q_j) \oplus I \oplus J$

Of course, if we restrict the language to have *only* guarded choice, then all the conflict names disappear in the axioms above, obtaining a rather simple net semantics for this sublanguage.

Finally, for the sake of simplicity we have presented the semantics for the monadic π–calculus, but it can be trivially extended to the polyadic calculus, where input and output carry a (possibly empty) tuple of channels ($x(y_1, \ldots, y_n).P$ and $\bar{x}\langle y_1 \ldots y_n\rangle.P$).

In case an input subprocess has an empty set of parameters, no generation of fresh names is required, hence its occurrence inside a recursion operator does not cause the generation of an infinite net.

From the observations above it trivially follows the following proposition that states a sufficient condition on p such that $Net(p)$ is finite.

Proposition 4.12 Let p be a process. If

(1) no restriction or unguarded sum occurs inside a recursion operator;
(2) p' is a derivative of p (i.e. $p \xrightarrow{a_1} \ldots \xrightarrow{a_n} p'$), $x \in fn(p')$ and $x(\bar{y}).q$ occurs in p' inside a recursion imply that $\bar{y}$ is the empty sequence;

then the subnet $Net(p)$ associated to p is finite. $\square$

The first condition states that the process p is generated by the following syntax:

$s ::= \mathbf{0} \mid \mu.t \mid s + s$
$t ::= s \mid t \,|\, t \mid X \mid recX.t$
$p ::= t \mid (\nu x)p \mid p \,|\, p \mid p + p$

where parallel composition may occur inside recursion. The second condition, even if not purely syntactic, is often verified in practical cases, e.g., for *closed systems* (i.e., processes where all the names are restricted). Let us call the π-calculus processes that satisfies the two conditions of Proposition 4.12, the *finite-net* processes. Note that finite-net processes may be infinite-state processes (i.e., the associated labeled transition system may be infinite).

Finally, we want to observe that in some cases the resulting net is actually a P/T net. This happens when considering closed systems (i.e., all actions are restricted) with guarded sum only. Indeed, in such a case, no conflict is generated by the decomposition function; moreover, as no extrusion is possible (only axioms (tau) and (sync) are applicable), all the restriction places (νx) can be safely removed. Similarly, if one wants to model only the reduction semantics [SW01] of a process, then only axioms (tau) and (sync) are applicable. An example of a non-trivial finite net process is reported Figure 9, where the resulting net is actually a P/T net because the π-calculus process is closed.

5 Non-interleaving Semantics

A step and a causal semantics for the π-calculus are induced from the corresponding semantics for PTI nets, recalled in Section 3. We illustrate these semantics by means of some examples.

5.1 Step Semantics

We present two examples, in order to clarify what cannot be performed in parallel. The system, according to the not optimized net semantics, corresponding to the process $\bar{a}x.\mathbf{0} + \bar{b}x.\mathbf{0}$ is depicted in Figure 4. Two transitions are enabled at the initial marking $m = \{k\bar{a}x.\mathbf{0}, \bar{k}\bar{b}x.\mathbf{0}\}$:

$$t_1 = (\{\{k\}\bar{a}x.\mathbf{0}\}, \{\bar{k}, (\nu a), (\nu x)\}) \xrightarrow{\bar{a}x} \{k\}$$

and

$$t_2 = (\{\{\bar{k}\}\bar{b}x.\mathbf{0}\}, \{k, (\nu b), (\nu x)\}) \xrightarrow{\bar{b}x} \{\bar{k}\}.$$

On the contrary, the step $\{t_1, t_2\}$ is not enabled at m, because $dom(t_1^\bullet) \cap {}^\circ t_2 = \{k\} \neq \emptyset$. Hence, the mechanism of distributed choice we have implemented is

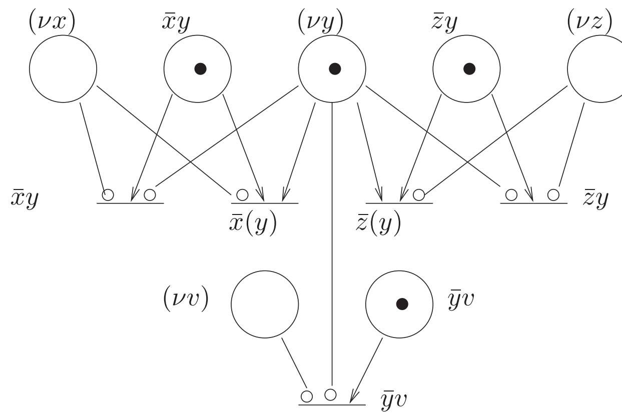

Fig. 5. The PTI system for $(\nu y)(\bar{x}y.\mathbf{0} \,|\, \bar{y}(v).\mathbf{0} \,|\, \bar{z}y.\mathbf{0})$.

compatible only with the step semantics of [BusPin99], and not with the more liberal step semantics of [JK95].

The PTI system corresponding to the process $(\nu y)(\bar{x}y.\mathbf{0} \,|\, \bar{y}v.\mathbf{0} \,|\, \bar{z}y.\mathbf{0})$ is depicted in Figure 5. Two transitions are enabled at the initial marking $m = \{(\nu y), \bar{x}y.\mathbf{0}, \bar{y}v.\mathbf{0}, \bar{z}y.\mathbf{0}\}$:

$$t_1 = (\{\bar{x}y.\mathbf{0}, (\nu y)\}, \{(\nu x)\}) \xrightarrow{\bar{x}(y)} \emptyset$$

and

$$t_2 = (\{\bar{z}y.\mathbf{0}, (\nu y)\}, \{(\nu z)\}) \xrightarrow{\bar{z}(y)} \emptyset.$$

The step $\{t_1, t_2\}$ is not enabled at m, because ${}^\bullet t_1 \oplus {}^\bullet t_2 \not\subseteq m$ (we do not have two tokens in the restriction place). On the contrary, in [MonPis95] the concurrent execution of t_1 and t_2 is allowed.

The step bisimulation between markings can be defined similarly to interleaving bisimulation in Definition 4.6 and, as expected, it is finer than the latter.

5.2 Causal Semantics

The causal semantics for PTI systems hinted in Section 3 and reported in more detail in the Appendix induces a (set of) causal semantics for π-calculus processes.

In [BorSan98,DegPri95], two kinds of causal dependencies between actions are singled out: the *structural* (or *subject*) dependencies, induced by the structure of processes, and the *link* (or *object*) dependencies, arising from the extrusion of restricted names. It is interesting to observe that these two kinds of causal relations closely correspond to the two different kinds of causal dependencies

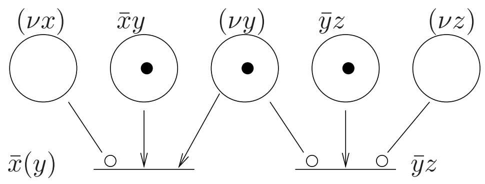

Fig. 6. The PTI system for $(\nu y)(\bar{x}y.\mathbf{0} \,|\, \bar{y}z.\mathbf{0})$

arising in PTI systems generated by a π-calculus term; in Definition A.8, two sets of immediate causes are defined: the set C^{flow} of causes due to flow arcs, containing all the causes due to the consumption of tokens produced by previously executed transitions, and the set C^{inib} of causes due to the presence of inhibitor arcs; a transition t is (inhibitor-) caused by all those transitions which removed tokens from the inhibiting places of t. The structural dependencies for π-calculus can be obtained by the transitive closure of the flow-causes, whereas the link dependencies correspond to the set C^{inib} of inhibitor-causes.

As an example of link dependency, consider the process $p = (\nu y)(\bar{x}y.\mathbf{0} \,|\, \bar{y}z.\mathbf{0})$, whose corresponding PTI system is drawn in Figure 6. This system can perform a causal transition labelled $\xrightarrow{\bar{x}(y),\emptyset,\emptyset}$, followed by the transition labelled $\xrightarrow{\bar{y}z,\emptyset,\{1\}}$. According to our definition, the second transition is (inhibitor-) caused by the first one, because the first removes the token that inhibited the second. Now consider process $q = (\nu y)(\bar{x}y.\bar{y}z.\mathbf{0})$. Process q can perform a causal transition labelled $\xrightarrow{\bar{x}(y),\emptyset,\emptyset}$, followed by the transition labelled $\xrightarrow{\bar{y}z,\{1\},\{1\}}$ which shows a double source of causality.

These processes are causal bisimilar or not, depending on what kind of comparison is made in the actual definition. p and q are mixed causal bisimilar, but neither distinct nor flow causal bisimilar (see Section 3 and the Appendix).

The causal bisimulation between markings can be defined similarly to interleaving bisimulation in Definition 4.6 and, as expected, mixed causal bisimulation is finer than step bisimulation.

6 π-calculus nets are primitive

We show that the PTI system $Net(p)$ generated by a π-calculus process p is primitive; hence, if it is also finite, it is possible to make use of the analysis techniques illustrated in Section 3.4 to study the behaviour of the process.

Proposition 6.1 For any π-calculus process p, the PTI system $Net(p)$ is primitive.

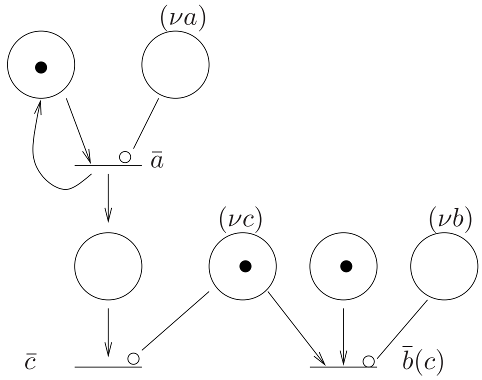

Fig. 7. The PTI system generated by process $(\nu c)(recX.(\bar{a}.(\bar{c}|X))|\bar{b}c)$.

Proof: The inhibiting places of $Net(p)$ are the conflict places k and the restriction places (νx). The emptiness limit for a conflict place is 0: in fact, no transition consumes tokens from such places, so once they are filled with a token, they cannot be emptied. A transition produces a token in a restriction place when, during the decomposition of a process, a restriction is encountered. The side condition of the decomposition function for a process with the form $(\nu x)p$ ensures that a "fresh" restriction place (i.e. never used) is utilized, so it is impossible to produce two tokens in the same place; then, the number of tokens is always bounded by 1. □

6.1 No reasonable semantics based on P/T nets

Here we give some evidence in support of the impossibility to provide the π-calculus with a sensible net semantics, based on standard P/T nets, which faithfully models the step behavior and the causal one.

First, we show that it is impossible, for a finite P/T system, to simulate the step behaviour of the primitive net generated by process

$$(\nu c)(recX.\bar{a}.(\bar{c}|X)|\bar{b}c)$$

which is finite and reported in Figure 7.

The step behaviour of this primitive system is essentially the same as the one of the system in Figure 3. The proof of the non existence of a finite P/T system simulating the step firing sequences of the primitive system in Figure 7 consists of a slight modification of the proof in [Bus02] of Theorem 3.9, consisting in replacing the labels a, b and c with $\bar{a}$, $\bar{b}(c)$ and $\bar{c}$, respectively. Hence, a P/T net simulating correctly the step firing sequences of net 7 has to be infinite.[4]

[4] Remember that if we are interested only in the interleaving semantics, then a

Moreover, the following argument tries to convince the reader that no physically realizable P/T net (i.e., a net with finite markings only and with transitions of finite synchronization only) can model correctly the causal behaviour of this net. As the action $\bar{b}(c)$ extrudes the name c, we have that all the occurrences of action $\bar{c}$ are link-caused (or inhibitor caused) by $\bar{b}(c)$; as the number of tokens that can be accumulated on place $\bar{c}$ is unbounded, and the only way to obtain a causal dependency between two transitions in a standard P/T system consists of making the first transition to produce a token which is consumed by the second one, to faithfully model the causal dependencies of this process by a P/T system it is necessary for transition $\bar{b}(c)$ to produce an infinite number of tokens, thus obtaining an unfeasible behaviour from a physical point of view.

6.2 Decidability results

As the PTI systems generated by π-calculus processes are primitive, the class of finite-net processes (as singled out in Proposition 4.12) enjoys some interesting properties, discussed in Section 3.4.

The coverability tree construction [KarMil69] has been extended in [Bus02] to finite primitive systems, thus permitting to decide some interesting properties. In particular, by inspection of the (finite) coverability tree associated to a finite primitive system one can see if, e.g., the net is bounded (in all reachable marking the number of tokens in each place is bounded by some constant), hence its behavior is finite-state. But also to check if one single place is bounded; this can be useful to reduce the size of a net: for example, if an inhibiting place is bounded by 0, we can safely remove the arcs exiting from that place. We can also see from the finite coverability tree if some sequential subsystem s is always active (in all the nodes of the tree s has value $n > 0$ or the special symbol ω that denotes unboundedness in the coverability tree construction); if in a choice some alternative subprocesses have never been chosen (it is enough to test that at least one conflict place corresponding to this alternative is always set to 0); if some restricted names are extruded or not (again, looking in the tree at the place corresponding to that restricted name). Another property following directly from the finiteness of the coverability tree is the existence of dead (i.e., that cannot be executed in any reachable marking) transitions. Indeed, it is enough to check that no arc in the coverability tree is labeled by that transition. Again, this may be useful in reducing the size of the net, as dead transitions can be safely removed.

The reachability problem (i.e., to check if a marking is reachable from the initial one) for finite primitive systems is decidable. This means that we can

finite P/T net simulating such a finite primitive net does exist.

search for (a finite set of) particular markings (e.g., those representing unsafe conditions) to see if they will ever be reached or not. Consequences of the decidability of reachability is that the liveness problem (a transition t is live if for any reachable marking m there exists a marking m' reachable from m such that t is enabled; a net is live if all of its transitions are live) as well as the deadlock problem (existence of a reachable marking m which is dead, i.e., no transition is enabled at m) are decidable. Hence, *total deadlock* is decidable; but also *partial deadlock* is decidable, because so is the the reachability problem for submarkings [Bus02]. [5]

By Theorem 3.11, we also get the decidability of the linear time μ-calculus for the class of finite-net π calculus processes. This is very interesting as the logic is quite expressive and the class of processes for which it is decidable is rather large, including many, practically relevant, infinite-state systems.

6.3 Two examples

Here we provide the net semantics of two simple, yet non-trivial, examples. The first one is the mobile telephones example reported in [Mil91]. The second one is an example of a producer-consumer system, where two producers send to a forwarder the products that two consumers will eventually consume. In both cases, the systems are finite-net processes: while the former is a finite-control [Dam96] system, the latter is not, because of the presence of parallel composition inside recursion in the forwarder process. Moreover, in both cases, as we are dealing with closed processes (all the names are restricted), no scope extrusion can be performed, thus we can safely remove all the inhibiting places and obtain a rather intuitive classic Place/Transition net system.

Let us now consider the mobile telephone example.

$$
\begin{aligned}
&(\nu talk_i, switch_i, give_i, alert_i, i = 1, 2)\\
&(\mathrm{rec}\, CAR.(talk_1.CAR + switch_1(talk_1, switch_1).CAR)|\\
&\mathrm{rec}\, BASE_1.(\overline{talk}_1.BASE_1 + give_1(t, s).\overline{switch}_1\langle t, s\rangle.alert_1.BASE_1)|\\
&\mathrm{rec}\, BASE_2.(\overline{talk}_2.BASE_2 + give_2(t, s).\overline{switch}_2\langle t, s\rangle.alert_2.BASE_2)|\\
&\mathrm{rec}\, CENTRE.(\overline{give}_1\langle talk_2, switch_2\rangle.\overline{alert}_2.\overline{give}_2\langle talk_1, switch_1\rangle.\\
&\overline{alert}_1.CENTRE))
\end{aligned}
$$

[5] The reachability problem for submarking is the following: Given a subset $S' \subseteq S$ of places and a marking m on S', there exists a marking $m' \in [m_0\rangle$ such that $m'(s) = m(s)$ for all $s \in S'$? The partial deadlock problem is the following: Given a subset $S' \subseteq S$ of places, there exists a reachable submarking m on S' that is dead?

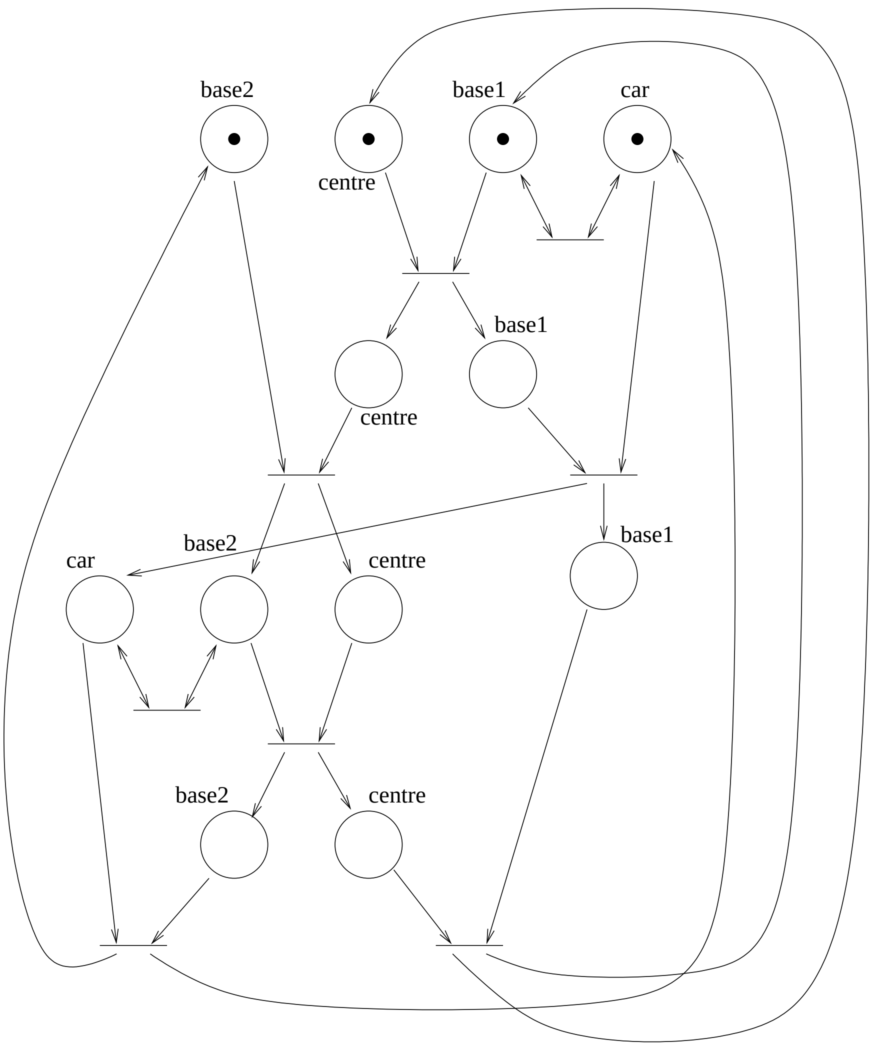

Fig. 8. The system corresponding to the mobile telephones example.

The net system corresponding to the process above is depicted in Figure 8. For simplicity sake, different places corresponding to the same sequential subprocess are called with the same name; e.g., we have four different places all called *centre*. The net clearly illustrates the connections between the various components, and the decision techniques developed for P/T systems can be used to prove e.g. that all transitions are live, or that in any reachable marking there is exactly one token in places corresponding to the centre (i.e. the centre does neither duplicate nor vanish). Of course, the net is also deadlock-free. This can be also seen by model checking: the formula $\nu Z.(\tau)Z$, which expresses that some progress (via a synchronization) is always possible, is satisfied.

Let us now consider the following producer-consumer system Sys:

$$Sys = (\nu a, b, c)(P_1|P_2|F|C_1|C_2)$$

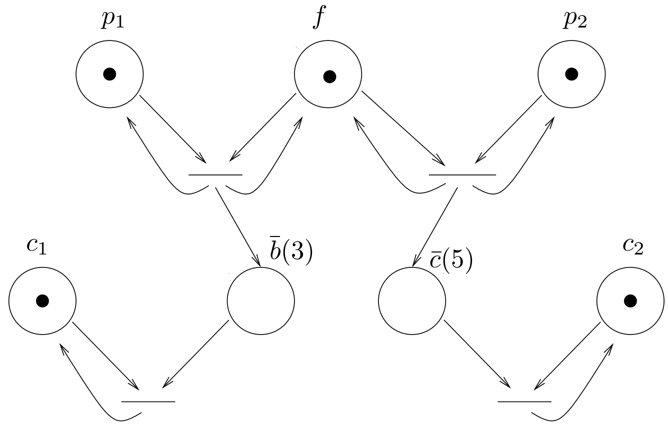

Fig. 9. The system corresponding to the producer-consumer example.

where P_1 and P_2 are producers and F forwards the requests to the consumers C_1 and C_2:

$$F = \mathrm{rec}\, X.(a(y,r).(X|\bar{r}(y)))$$

$$P_1 = \mathrm{rec}\, X.(\bar{a}(3,b).X) \qquad P_2 = \mathrm{rec}\, X.(\bar{a}(5,c).X)$$

$$C_1 = \mathrm{rec}\, X.(b(w).X) \qquad C_2 = \mathrm{rec}\, X.(c(w).X)$$

The finite net system corresponding to Sys is depicted in Figure 9, where place $p_1 = \bar{a}(3,b).\mathrm{rec}\, X.(\bar{a}(3,b).X)$, place $p_2 = \bar{a}(5,c).\mathrm{rec}\, X.(\bar{a}(5,c).X)$, place $f = a(y,r).(\mathrm{rec}\, X.(a(y,r).(X|\bar{r}(y))) \mid \bar{r}(y))$, place $c_1 = b(w).\mathrm{rec}\, X.(b(w).X)$ and, finally, place $c_2 = c(w).\mathrm{rec}\, X.(c(w).X)$. Observe that, because of the asynchronous behaviour of producers and consumers, the number of tokens that may be present in places $\bar{b}(3)$ and $\bar{c}(5)$ can be unbounded. Indeed, process Sys is an infinite-state system.

7 Further Issues

7.1 *Related Work*

Net Semantics
The first paper dealing with a net semantics is [Eng96]. There Engelfriet considers the small π-calculus (no alternative composition), for which he studies the reduction semantics (communications only). Because of these limited aims, the semantics is greatly simplified: the restriction operator can always be syntactically removed by simply renaming bound actions to fresh names. Hence, there is no need for inhibitors. It is easy to see that our net semantics

conservatively extends his proposal. [6]

A recent paper is [DKK06a], where Devillers, Klaudel and Koutny define a more complex semantics for finite (i.e., recursion-free) π-calculus using high-level nets with read arcs. The main feature of their approach is that finite nets are obtained in a purely compositional way by adapting existing graph-theoretic net composition operators, by first translating a process into a context-based representation that removes restrictions. Their interesting work has been subsequently extended in [DKK06b] to take care also of recursive behaviours by using techniques used in the context of PBC [BDK]. Surprisingly enough, their approach produces finite high-level nets with read arcs also for (even unguarded) recursive terms, at the price, however, of admitting places with infinitely many tokens. Even if not explicitly worked out in their papers, their approach could also permit the definition of concurrent semantics for π-calculus.

An interesting P/T net semantics for a variant of π-calculus – where summation is always guarded, recursion is modelled by a step of unwinding and only reduction semantics [SW01] is considered – is studied in [Mey07]. There Meyer shows that the interleaving marking graph of the net associated to a process p is isomorphic to the transition system defined by the reduction semantics of p. His primary goal is to obtain minimal net realizations of the reduction semantics, in order to apply all the set of verification techniques available for finite P/T nets. He singles out a large class of π-calculus processes, the so-called *structurally stationary* processes, for which his net semantics is finite. This class of processes is incomparable with the class of finite-net processes we single out for our PTI semantics. For instance, the process Sys of Section 6.3 (as well as process $(\nu a)\text{rec}\, X.(\bar{a}(w) \mid \tau.X))$ is finite-net but not structurally stationary; on the contrary, process $\text{rec}\, X.(\nu a)(\bar{a}(w) \mid \tau.X)$ is not finite-net, but it is structurally stationary. Note that his results are not in contrast with our claim that it is impossible to provide a faithful P/T net semantics to π-calculus for the following reasons: (i) If we consider only the reduction semantics, as Meyer does, we have already observed that our net semantics also generates (not necessarily finite) P/T nets. (ii) Meyer's net construction does not preserve the intended non-interleaving semantics of π-calculus processes, at least for the way restriction is handled. For instance, in process $(\nu c)(a.c \mid b.\bar{c}) \mid (\bar{a} \mid \bar{b})$ the two reductions on a and b are causally dependent in his semantics. Note also that, if we restrict our attention to reduction semantics only, our net semantics can be used to provide, sound (both w.r.t. interleaving and causal semantics), finite net P/T representation for the class

[6] To be precise, this is true with the only exception of the bang operator (i.e., replication), as we prefer to use guarded CCS-like recursion in order to avoid the problem of managing markings composed of infinitely many places and/or infinitely many tokens in a place (see Section 7.2 for more details).

of finite-net processes, hence also for many, practically relevant, infinite-state systems.

Causal Semantics

The first causal semantics was proposed in [BorSan98], where Boreale and Sangiorgi define an early causal transition system. They observe that there are two different sources of causality: the *subject* (or structural) dependencies, due to the syntactical structure of terms (causes produced by the nesting of prefixing or inherited through communications), and the *object* (or link) dependencies, due to the order in which names are used in a computation (a name introduced in a transition labeled by μ_j is then among the free names of a later transition labeled μ_i: they say that μ_i is object dependent on μ_j). For instance, in the (interleaving) run $(\nu b)(b.\mathbf{0} \mid a(x).\bar{x}b.\mathbf{0}) \xrightarrow{ac} \xrightarrow{\bar{c}(b)} \xrightarrow{b} \mathbf{0} \mid \mathbf{0}$, the second action is object dependent on the first action, as well as the third is object dependent on the second. Object dependencies can be observed already at the interleaving level and so, as their goal is to define a fully abstract encoding of the causal transition system over the standard interleaving one, they are ignored in the causal transition system.

Further work is necessary to properly compare the two approaches, also because their causal bisimulation is a weak one, while ours is a strong one. However, we conjecture that our flow causality is essentially the same relation as their subject causality. Differently from their approach, we also explicitly model those object dependencies due to extrusion, which enables transitions which were previously blocked by the presence of restriction (in the example above, we have that the third action is inhibiting caused by the second action, but the second action is only flow caused by the first one). Hence, the inhibiting causes are a proper subset of the object dependencies. However, the object dependencies not captured by inhibiting causes are already captured by the flow causes.

On the one hand, only the class of structural/subject causes is explicitly described in their transition system. On the other hand, our mixed causal semantics, instead, joins together the causes originated by both classes of phenomena. Consider the two processes $p = (\nu y)(\bar{x}y.\mathbf{0} \mid \bar{y}z.\mathbf{0})$ and $q = (\nu y)(\bar{x}y.\bar{y}z.\mathbf{0})$. These processes are not causal bisimilar according to [BorSan98] (there is a subject dependency in q), whereas they are mixed causal bisimilar for us (see also Section 5.2).

Our approach to causality is also different from the one given in [DegPri95] [DegPri99]. First of all, Degano and Priami follow the so-called *read-write* causality, according to which outputs can pass their causes to inputs but not vice versa (i.e., the cause-crossing in communication is not symmetric but directioned from outputs to inputs). The "symmetric" causes missing in communication are classified as structural priority. Similarly for link dependencies, they define also object priority. Further work is necessary to compare our work to theirs. However, we conjecture that the reflexive closure of our mixed causal semantics coincides with the enabling relation of [DegPri95], which roughly corresponds to the transitive and reflexive closure of the union of the four relations above (read-write causality, structural priority, link causality, object priority).

In [MonPis95] Montanari and Pistore propose a graph–rewriting system (based on the double pushout approach) as a semantic model for a subset of π-calculus, where $+$ is omitted and the bang operator is applied only to input prefixing. Apart from the technical differences, the basic intuition about restriction is similar to ours, even if "complemented": a "token" in a "place" for x means that x is free; hence they test for presence of a token in the "place" for x. However, this causes the debatable phenomenon of parallel extrusion. Consider the process $p = (\nu y)(\bar{x}y.\mathbf{0} \,|\, y(w).\mathbf{0} \,|\, \bar{z}y.\mathbf{0})$: the two actions $\bar{x}(y)$ and $\bar{z}(y)$ are considered independent and executable in parallel in [MonPis95], while in interleaving semantics only one of the two outputs is actually a bound output. This example is also discussed in Section 5.1, where we argue that in our semantics this parallel extrusion is not possible.

Another paper is [JatJag95], where Jategaonkar and Jagadeesan propose a data-flow semantics for π-calculus, based on dI-domains. As the notion of causality they use is classical, it seems likely that it coincides with our mixed causal approach, even if in a trace-based setting. Further study is anyway necessary to substantiate the claim.

Finally, Cattani and Sewell in [CatSew00] define a syntax-free causal model for name-passing processes in terms of indexed labelled asynchronous transition systems. Their goal is, in some sense, similar to ours, as we also provide a causal semantics to π calculus that is model-driven (actually, ours [BusGor95] is the first model-driven semantics for the π calculus). Also in this case, further work is needed to set properly a formal correspondence between the two approaches. Nonetheless, it seems that their *name-dependency aware history preserving bisimulation* is the weak version of our mixed causal bisimulation.

Decidability Results

We have shown the decidability of the satisfiability problem for formulae of the linear time μ-calculus for the class of *finite-net* processes, i.e., those processes where restriction and unguarded sum cannot occur inside a recursion. Note that this class is incomparable with the one of *finite-control* processes (obtained by preventing parallel composition to occur inside recursion), for which the modal μ-calculus is shown to be decidable in [Dam96].

The net semantics for the variant of π-calculus studied by Meyer in [Mey07] permits to provide finite P/T net representation for the class of so-called *struc-*

turally stationary processes (a class strictly larger than Dam's finite-control processes, but incomparable with our finite-net processes). Hence, all the decidability results that hold for finite P/T nets are inherited by π-calculus processes in this class. In [MKS08] the net semantics proposed in [Mey07] is used to develop a practical, more efficient model-checking technique for the class of finite-control processes. One further recent paper by Meyer [Mey08] studies a superclass of both structurally stationary and finite-net processes, the so-called *bounded in depth*, for which he shows that the reduction semantics generates a transition system onto which it is possible to single out a clever well-quasi-ordering on the states (processes) which is compatible with the reduction relation. This permits to prove that termination is decidable. (Note that the deadlock problem (hence, termination) is decidable for finite primitive nets – proof in [Bus02].)

7.2 Matching and Bang Operators

The matching operator can be easily accommodated in our approach. The decomposition function is extended with the following clause:

$$dec([x = y]p) = \begin{cases} dec(p) & \text{if } x = y \\ \mathbf{0} & \text{otherwise} \end{cases}$$

and there is no need of any additional axiom for transitions. Indeed, it is treated as simple syntactic sugar, and has no impact on finiteness of the net semantics, nor on decidability issues.

The bang operator, often called *replication*, can be modeled with some difficulty. A naive solution could be to take the fixed-point of the recursive equation:

$$dec(!p) = dec(p) \cup dec(!p)$$

There are two possible outcomes. If in p there are no occurrences of restrictions or (unguarded) sums, then the solution marking is composed of infinitely many tokens on the places of $dec(p)$. Instead, if at least one of these operators occurs in p, then the (domain of the) solution marking has infinitely many places. In both cases, the solution seems unpractical (it is not physically realizable), and this is the reason why we have preferred CCS-like guarded recursion. Note that the first outcome is essentially the solution proposed in [Eng96].

Note also that we cannot escape this problem even in case we restrict the bang to (input) prefixes only. As a matter of fact, we cannot treat $!\mu.p$ as

$recX.(\mu.(p \mid X)$ because the two terms should have different causal semantics.

For the case in which p has no occurrences of restrictions and sums, there is another solution which, in our opinion, is better. It is implementable using read arcs. Transitions are then of the form $(c, i, r) \xrightarrow{a} p$, where the additional set r denotes the set of places which are to be "read", i.e., tested for presence. The decomposition of $!p$, then, can be defined as follows:

$$dec(!p) =!dec(p)$$

where ! is a decoration of the sequential processes originated by $dec(p)$. A transition from a place in $!dec(p)$ is always possible with no consumption of tokens. For instance, a transition for the process $!\bar{x}y.p$ is the following:

$$(\emptyset, \{(\nu x), (\nu y)\}, \{!\bar{x}y.p\}) \xrightarrow{\bar{x}y} dec(p)$$

where no token is consumed, (νx) and (νy) are tested for absence and $!\bar{x}y.p$ is tested for presence. This solution is essentially the one proposed in [MonPis95] in the context of graph-grammars.

7.3 Conclusion and future work

Primitive nets (i.e., P/T nets with inhibitor arcs where a certain bound can be computed for the inhibiting places such that if, during the token game, that bound is overcome, that place cannot be emptied anymore) are used to give a distributed semantics to a rich dialect of π-calculus. The advantage is that, when the net associated to π-calculus process is finite, some interesting properties can be decided, notably satisfiability of formulae of the linear time μ-calculus. The fragment of so-called *finite-net* π-calculus is rather large, admitting parallel composition inside recursion. The soundness of the semantics is showed by providing a fully abstract result: two processes p and q are early interleaving bisimilar if and only if the net markings $dec(p)$ and $dec(q)$ are early interleaving bisimilar.

We also showed that, if we restrict our attention to reduction semantics, the resulting net associated to π-calculus processes is a standard (not necessarily finite) P/T net. This is quite useful because finite-net processes generate finite P/T net model over which many well-known properties are decidable.

Primitive nets are also equipped with non-interleaving semantics (step and causal), hence these can be transferred to π-calculus processes as well. Those reported here (based on [BusGor95]) are the first non-interleaving semantics for the π-calculus that were not defined syntactically, rather derived from a model.

Some arguments are given to prove that primitive nets represents the most elementary class of nets that can be used to give distributed semantics to π-calculus. In particular, we show that it is not possible to provide physically realizable (i.e., with finite markings and with transitions of finite synchronization) P/T net semantics respecting the intended non-interleaving semantics.

Future work may be devoted to the study of further optimization techniques that may be helpful in reducing the size of the resulting nets, so that larger fragments of the π-calculus may be given a finite primitive net representation. For instance, one can make an explicit handling of the new names, according to some clever discipline. Following ideas on the reuse of obsolete names of, e.g., [MonPis97], one can define a net semantics that may be more compact than the one defined in this paper and that, in some cases, offers a finite primitive model also for some π processes where restriction lies inside recursion. Other future work may include:

- the study of the complexity of the decision procedure for the various decidable properties over finite primitive nets;
- the realization of a software tool that implements such decision procedures;
- the use of existing tools for finite P/T nets to analyze closed, finite-net π-calculus processes;
- the study of truly concurrent (linear time) logics, e.g., logics where the basic action is replaced by, e.g., a step of actions. As a matter of fact, it is possible to define a linear time *step* μ-calculus, where the basic interleaving operator $\langle a \rangle\phi$ is replaced by $\langle A \rangle\phi$ (where A is a multiset of actions), and we conjecture that the model checking problem for the linear time step μ-calculus and finite primitive systems is decidable. This would offer a logic for checking some non-interleaving properties also for π-calculus processes.

Acknowledgements. The second author would like to thank Davide Sangiorgi and Roland Meyer for helpful suggestions and comments. Also the two anonymous referees are thanked for their useful suggestions. The first draft of this paper was written around the end of year 2000. The second author has collected that material after first author's premature death in September 2007, and updated the paper with relevant, more recent, related work.

References

[Age74] T. Agerwala, "A complete model for representing the coordination of asynchronous processes", Hopkins Computer Research Report 32, John Hopkins University, 1974.

[BerBou92] G. Berry, G. Boudol, "The Chemical Abstract Machine", *Theoretical Computer Science* 96(1):217-248, 1992.

[BDK] E. Best, R. Devillers, M. Koutny, "A unified model for nets and process algebras", Chapter 14 of *Handbook of Process Algebra*, 873-944, Elsevier, 2001.

[BorSan98] M. Boreale, D. Sangiorgi, "A fully abstract semantics of causality in the π-calculus", *Acta Informatica* 35(5):353-400, 1998. (An extended abstract appeared in Proc. *STACS'95*, LNCS 900, 243-254,Springer, 1995).

[Bus02] N. Busi, "Analysis Issues in Petri Nets with Inhibitor Arcs", *Theoretical Computer Science* 275(1-2):127-177, 2002.

[BusGor94] N. Busi, R. Gorrieri, "Distributed Conflicts in Communicating Systems", in *Object-Based Models and Languages for Concurrent Systems*, LNCS 924, 49-65, Springer, 1994.

[BusGor95] N. Busi, R. Gorrieri, "A Petri Net Semantics for π- calculus", In Proc. *Concur'95* , LNCS 962, Springer, 145-159, 1995.

[BusPin99] N. Busi, G. M. Pinna, "Process Semantics for Place/Transition Nets with Inhibitor and Read Arcs", *Fundamenta Informaticae* 40(2-3):165-197, 1999.

[BusPin00] N. Busi, G. M. Pinna, "Comparing Truly Concurrent Semantics for Contextual Place-Transition Nets", *Fundamenta Informaticae* 44(3):209-244, 2000.

[CatSew00] G.L. Cattani, P. Sewell, "Models for Name-passing Processes: Interleaving and Causal", in Proc. *LICS'00*, 322-332, IEEE Press, 2000.

[CEP95] A. Cheng, J. Esparza, J. Palsberg, "Complexity results for 1-safe nets", *Theoretical Computer Science* 147:117-136, 1995.

[Dam96] M. Dam, "Model Checking Mobile Processes", *Information and Computation* 129(1):35-51, 1996. (An extended abstract appeared in Proc. *CONCUR'93*, LNCS 715, 22-36,Springer, 1993).

[DarDeg89] Ph. Darondeau, P. Degano, "Causal Trees", in Proc. *ICALP'89*, LNCS 372, 234-248, Springer, 1989.

[DDM88] P. Degano, R. De Nicola, U. Montanari, "A Distributed Operational Semantics for CCS based on C/E Systems", *Acta Informatica* 26(1-2):59-91, 1988.

[DGM88] P. Degano, R. Gorrieri, S. Marchetti, An Exercise in Concurrency: A CSP Process as a Condition/Event System, *Advances in Petri Nets 1988*, LNCS 340, 85-105, Springer, 1988.

[DegPri95] P. Degano, C. Priami, "Causality for Mobile Processes", in Proc. *ICALP'95*, LNCS 944, Springer, 660-671, 1995.

[DegPri99] P. Degano, C. Priami, "Non Interleaving Semantics for Mobile Processes", *Theoretical Computer Science*216(1-2):237-270, 1999.

[DKK06a] R. Devillers, H. Klaudel, M. Koutny, "Petri Net Semantics of the Finite pi-calculus Terms", *Fundamenta Informaticae* 70(3):203-226, 2006. (An extended abstract appeared in Proc. *FORTE'04*, LNCS 3235, 309-325,Springer, 2004).

[DKK06b] R. Devillers, H. Klaudel, M. Koutny, "A Petri Net Translation of pi-Calculus Terms", in Proc. *ICTAC'06*, LNCS 4281, 138-152,Springer, 2006.

[Eng96] J. Engelfriet, "A Multiset Semantics for the π-calculus with Replication", Theoretical Computer Science153(1-2):65-94, 1996. (An extended abstract appeared in Proc. *CONCUR'93*, LNCS 715, Springer, 7-21, 1993).

[Esp94] J. Esparza, "On the decidability of model-checking for several mu-calculi and Petri nets", in Procs. *CAAP'94*, LNCS 787, 115-129, Springer, 1994.

[Esp97] J. Esparza, "Decidability of model-checking for infinite-state concurrent systems", *Acta Informatica* 34:85-107, 1997.

[GlaVaa87] R. van Glabbeek, F. Vaandrager, "Petri Net Models for Algebraic Theories of Concurrency", in Proc. *PARLE'87*, LNCS 259, 224-242, Springer, 1987.

[Gol88] U. Goltz, "On Representing CCS Programs by Finite Petri Nets", in Proc. *MFCS'88*, LNCS 324, 339-350, Springer, 1988.

[Gol90] U. Goltz, "CCS and Petri Nets", LNCS 469, 334-357, Springer, 1990.

[GolRen94] U. Goltz, A. Rensink, "Finite Petri nets as models for recursive causal behaviour", *Theoretical Computer Science* 124(1):169-179, 1994.

[GorMon90] R. Gorrieri, U. Montanari, "SCONE: A Simple Calculus of Nets", in Proc. *CONCUR'90*, LNCS 458, 2-30, Springer, 1990.

[GorMon95] R. Gorrieri, U. Montanari, "On the Implementation of Concurrent Calculi in Net Calculi: Two Case Studies", *Theoretical Computer Science* 141(1-2):195-252, 1995.

[Hack74] M. Hack, "The recursive equivalence of the reachability problem and the liveness problem for Petri nets and Vector Addition Systems", in Procs. 15th Ann. Symp. on Switching and Automata Theory, IEEE, 1974.

[Hack76] M. Hack, "Petri Net Languages", Technical Report 159, MIT, 1976.

[JK95] R. Janicki, M. Koutny, "Semantics of Inhibitor Nets", *Information and Computation* 123:1-16, 1995.

[JatJag95] L. Jategaonkar Jagadeesan, R. Jagadeesan, "Causality and True Concurrency: A Data-flow Analysis of the Pi-Calculus", in Proc. *AMAST'95*, LNCS 936, 277-291, Sprimger, 1995.

[KarMil69] R.M. Karp, R.E. Miller, "Parallel Program Schemata", *Journal of Computer and System Sciences* 3(2):147-195, 1969.

[Kos82] S.R. Kosaraju, "Decidability of reachability in Vector Addition Systems", in Procs. 6th ACM STOC, 267-281, ACM Press, 1982.

[Koz83] D. Kozen, "Results on the Propositional mu-calculus", *Theoretical Computer Science* 27:333-354, 1983.

[May84] E.W. Mayr, "An algorithm for the general Petri net reachability problem", *SIAM J. Comput.* 13:441-460, 1984.

[Mey07] R. Meyer. "A Theory of Structural Stationarity in the π-Calculus", to appear in *Acta Informatica*, 2007.

[Mey08] R. Meyer. "On Boundedness in Depth in the π-Calculus", In Procs. 5th IFIP International Conference on Theoretical Computer Science, to appear.

[MKS08] R. Meyer, V. Khomenko, T. Strazny, "A Practical Approach to Verification of Mobile Systems Using Net Unfoldings", in Procs. ATPN'08, LNCS 5062, 327-347, 2008.

[Mil89] R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.

[Mil91] R. Milner, "The Polyadic π–Calculus: A Tutorial", Technical Report, Department of Computer Science, University of Edinburgh, ECS–LFCS–91–180, October 1991.

[MPW92] R. Milner, J. Parrow, D. Walker, "A Calculus of Mobile Processes", *Information and Computation* 100(1), 1-77, 1992.

[MonPis95] U. Montanari, M. Pistore, "Concurrent Semantics for the π-calculus", in Proc. MFPS'95, ENTCS 1, Elsevier, 1995.

[MonPis97] U. Montanari, M. Pistore, "Minimal Transition Systems for History-Preserving Bisimulation", in Proc. STACS'95, LNCS 1200, 413-425, Springer, 1997.

[MonRos95] U. Montanari, F. Rossi, "Contextual nets" *Acta Informatica* 32:545-596, 1995.

[Old91] E. R. Olderog, *Nets, Terms and Formulas*, Cambridge Tracts in Theoretical Computer Science 23, CUP, 1991.

[RabTra88] A. Rabinovich, B. A. Trakhtenbrot, "Behaviour structures and nets", *Fundamente Informaticae* 11:357-404, 1988.

[Rei85] W. Reisig, *Petri Nets: An Introduction*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.

[SW01] D. Sangiorgi, D. Walker, *The π-calculus: A Theory of Mobile Processes*, Cambridge University Press, 2001.

[Vog97] W. Vogler, "Partial Order Semantics and Read Arcs", in Proc. MFCS'97, LNCS 1295, 508-517, Springer, 1997.

A Causal semantics for PTI Systems

We recall the formal definitions for the causal, mixed ordering semantics for PTI systems as proposed in [BusPin00]. We start by defining the causal semantics on P/T systems, then we will extend it to cope with inhibitor arcs.

Definition A.1 Let $N = (S, T, F, m_0)$ be a P/T system.The set of *token types* is $\Theta = (T \times \omega^+) \cup \{*\}$, ranged over by θ, where (t, i) is the type of tokens produced by the i-th occurrence of transition t and $*$ is the type of tokens in the initial marking. A *configuration* γ of a net is a pair (p, o), where

- $p : S \to \Theta \to \omega$ describes for each place the number of tokens of each type it contains;
- $o : T \to \omega$ associates to each transition the number of times it has fired, i.e. the number of occurrences of the transition in the computation.

The *initial configuration* of the net is $\gamma_0 = (p_0, o_0)$, where

$$p_0(s)(\theta) = \begin{cases} m_0(s) & \text{if } \theta = * \\ 0 & \text{otherwise} \end{cases} \qquad\qquad o_0(t) = 0 \qquad \square$$

With $\gamma[(t, i), C\rangle\gamma'$ we denote the firing of transition t from configuration γ to configuration γ'. Actually, it is the i-th time that transition t is fired. Set C records the immediate causes for the firing of this occurrence; its elements are occurrences of transitions, i.e. elements of $T \times \omega^+$. We extend the labelling function l to occurrences of transitions in the following way: $l(t, i) = l(t)$ and τ ranges over $T \times \omega^+$.

Definition A.2 The rule for the *i-causal firing rule* is as follows: [7]

$(p, o)[(t, i), C\rangle(\bar{p}, \bar{o})$ if and only if

- $\exists p' \subseteq p$ such that, for all $s \in S$, ${}^\bullet t(s) = \sum_\theta p'(s)(\theta)$
- $i = o(t) + 1$
- $C = \{\tau \mid \exists s : p'(s)(\tau) > 0\}$
- $\bar{p} = (p \setminus p') \oplus p''$, where $p''(s)(\theta) = \begin{cases} t^\bullet(s) & \text{if } \theta = (t, i) \\ 0 & \text{otherwise} \end{cases}$
- $\bar{o}(u) = \begin{cases} i & \text{if } u = t \\ o(u) & \text{otherwise} \end{cases}$ $\square$

[7] Note that $S \to \Theta \to \omega$ is isomorphic to $(S \times \Theta) \to \omega$, so we can treat its elements as multisets.

The multiset p' denotes the set of decorated tokens consumed by transition t; the requirement $p' \subseteq p$ ensures that there are enough tokens for t to fire in each place s. Note that in general there exist more than one p' satisfying the condition (but a finite number of such p's, as p is a finite multiset): in fact, only the number of tokens we have to pick from a given place is fixed, but their decoration is not; the choice of p' may influence the definition of the set of immediate causes C. The occurrence number i associated to t is obtained by incrementing by one the number of the previous firings of t, stored in $o(t)$. The set of immediate causes C of the transition occurrence (t, i) corresponds to the set of transition occurrences that produced the tokens consumed by (t, i); the information about the transition occurrence that produced a token is stored in the token decoration; the tokens already present in the initial marking are decorated with $*$, hence do not contribute to incrementing the set of immediate causes. We update the configuration of the net by removing the consumed tokens (represented by p'), and adding the multiset p'' of the tokens produced by (and decorated with) (t, i); moreover, we update the counter of transition firings by incrementing by one the counter of transition t.

Definition A.3 An *i-causal firing sequence* (CFS) is defined inductively as follows:

- γ_0 is a CFS;
- if $\gamma_0[(\tau_1, C_1\rangle\gamma_1 \ldots [\tau_{n-1}, C_{n-1}\rangle\gamma_{n-1}$ is a CFS and $\gamma_{n-1}[\tau_n, C_n\rangle\gamma_n$ then $\gamma_0[\tau_1, C_1\rangle\gamma_1 \ldots [\tau_{n-1}, C_{n-1}\rangle\gamma_{n-1}[\tau_n, C_n\rangle\gamma_n$ is a CFS.

The set of i-causal firing sequences of a net N is denoted by $CFS(N)$. $\square$

Definition A.4 Let $\gamma_0[(\tau_1, C_1\rangle\gamma_1 \ldots [\tau_n, C_n\rangle\gamma_n$ be a CFS. We define the relation $\lhd$ as $\tau_i \lhd \tau_j$ iff $\tau_i \in C_j$.

Note that if $\tau_i \lhd \tau_j$ then $i < j$; note also that $\lhd^+$ is a (strict) partial order.

Causal trees [DarDeg89] are trees labelled with pairs (a, I), where a is an action and I is a set of relative pointers to all the predecessors which caused the present action a. We associate a causal tree to the set of the i-causal firing sequences $CFS(N)$.

Definition A.5 Let N be a labelled P/T system. The causal tree of N is the tree $CT(N) = (V, A, \gamma_0)$ defined as follows:

- $V = CFS(N)$
- $A = \{\sigma \xrightarrow{a,I} \sigma[\tau_n, C_n\rangle\gamma_n \mid \sigma = \gamma_0[\tau_1, C_1\rangle\gamma_1 \ldots [\tau_{n-1}, C_{n-1}\rangle\gamma_{n-1} \wedge a = l(\tau_n)\} \wedge$ $I = \{n - i \mid \tau_i \lhd^+ \tau_n\}$ $\square$

Definition A.6 Two P/T systems N_1 and N_2 are *causal bisimilar* ($N_1 \sim_c N_2$) iff there exists a bisimulation between $CT(N_1)$ and $CT(N_2)$. $\square$

Moving to PTI systems, besides the causal relation due to the flow arcs, a new kind of causes arises, due to inhibitor arcs. If we have an inhibitor arc (s, t), then an occurrence of t can fire only if place s is empty (i.e. contains no tokens); so, all the events that consumed tokens from s are inhibitor-causes for the occurrence of t; hence, we need to extend the definition of configuration, by recording for each place the set of events which consumed tokens from it.

Definition A.7 Let $N = (S, T, F, I, m_0)$ be a PTI system. A *configuration* γ of a net is a triple (p, c, o), where

- p and o are defined as in Definition A.2.
- $c : S \to \wp(T \times \omega^+)$ describes for each place the set of (occurrences of) transitions which have consumed tokens from that place.

The *initial configuration* of the net is $\gamma_0 = (p_0, c_0, o_0)$, where p_0 and o_0 are defined as in Definition A.2 and $c_0(s) = \emptyset$ for all $s \in S$. □

Definition A.8 The rule for the i-causal firing rule is as follows: $(p, c, o)[(t, i), C^{flow}, C^{inib}\rangle(\bar{p}, \bar{c}, \bar{o})$ if and only if

- $\exists p' \subseteq p$ such that for all $s \in S$ $\ {}^\bullet t(s) = \sum_\theta p'(s)(\theta)$
- $\forall s \in {}^\circ t \ \ \sum_\theta p(s)(\theta) = 0$
- $o(t) = i - 1$
- $C^{flow} = \{\tau \mid \exists s : p'(s)(\tau) > 0\}$
  $C^{inib} = \bigcup_{s \in {}^\circ t} c(s)$
- $\bar{p} = (p \setminus p') \oplus p''$, where $p''(s)(\theta) = \begin{cases} t^\bullet(s) & \text{if } \theta = (t, i) \\ 0 & \text{otherwise} \end{cases}$
- $\bar{c}(s) = \begin{cases} c(s) \cup \{(t, i)\} & \text{if } s \in {}^\bullet t \\ c(s) & \text{otherwise} \end{cases}$
- $\bar{o}(u) = \begin{cases} i & \text{if } u = t \\ o(u) & \text{otherwise} \end{cases}$ □

The set of flow-causes C^{flow} is constructed as the set of causes for P/T systems. The set of inhibitor-causes C^{inib} of (t, i) corresponds to the set of events that removed tokens from the inhibiting places of t; this information is stored, for each place, by c. We update the function c by adding the event (t, i) to each set $c(s)$ corresponding to a place s from which t consumes tokens.

The set $CFS(N)$ of i-causal firing sequences of the PTI system N can be given also for this enriched setting as done above for P/T nets, with the only difference that two sets of causes, C^{flow} and C^{inib}, are reported in each transition, instead of a single C. A CFS induces two dependency relations: $\tau_i \lhd_{flow} \tau_j$ iff $\tau_i \in C_j^{flow}$ and $\tau_i \lhd_{inib} \tau_j$ iff $\tau_i \in C_j^{inib}$. At this point, we could make no distinction between the various kinds of causality and construct a causal tree in the same way as for P/T system; in this case, we define $\lhd = \lhd_{flow} \cup \lhd_{inib}$ and we take the transitive closure $\lhd^+$ to determine the correct naturals in the labels of the tree. Or we can keep the two sets of causes separated; in this case, the causal tree would be decorated by triple (a, I, J), where I is generated by means of the relation $\lhd^+_{flow}$ and J by means of the relation $\lhd^+_{inib}$. We prefer the latter approach because the resulting causal tree can be used in a more flexible way to compare different sources of causality.

Two PTI systems N_1 and N_2 are *causal bisimilar* ($N_1 \sim_c N_2$) iff there exists a bisimulation between $CT(N_1)$ and $CT(N_2)$. Actually, three different versions of causal bisimulation can be defined: (i) *mixed causal*: when in the bisimulation game we compare (a, I, J) with (a, I', J'), pretending that $I \cup J = I' \cup J'$. (ii) *distinct causal*: when in the bisimulation game we compare (a, I, J) with (a, I', J'), pretending that $I = I'$ and $J = J'$. (iii) *flow causal*: when in the bisimulation game we compare (a, I, J) with (a, I', J'), pretending that $I = I'$ and ignoring J, J'. The flow causal is actually the approach of [BorSan98], where link/object causes are ignored because interleaving equivalent processes show the same object dependencies.

In [BusPin99] a process semantics for PTI systems has been proposed, and [BusPin00] showed that this is equivalent to the (mixed) causal semantics.